- RowInit
- SearchInit

FieldDefault

The event is triggered before the panel is displayed to the user. As discussed in part 2, field values can be defaulted in the Application Designer and can also be defaulted from the FieldDefault event. The Application Processor examines the fields in a panel group and, if the field is blank (character field) or zero (numeric field), the Application Processor sets the field to any value specified in the Application Designer Record Field Properties. When no default value is specified, any FieldDefault PeopleCode associated with the field is executed. It is important to note that, if a field has been assigned a default value in the Record Field Properties, and we also happen to include a neat little piece of FieldDefault PeopleCode, the Application Processor determines if the field is blank or zero before any FieldDefault program execution. Because the default value from the record definition already filled the field, the Application Processor does not execute the FieldDefault PeopleCode.

FieldDefault is an iterative event that constantly checks for blank (character field) or zero (numeric field) in the panel group. When a field does not have a default value in its Record Field Properties and does not have a value as a result of data entry or some other PeopleCode event, the FieldDefault event executes as a result of another event occurring. The other events can include fields on a panel. Figure 13.1 is an example of FieldDefault PeopleCode used in the Problem Tracking application. Note that more than one field can be defaulted. This enables us to consolidate code rather than spreading it across fields.

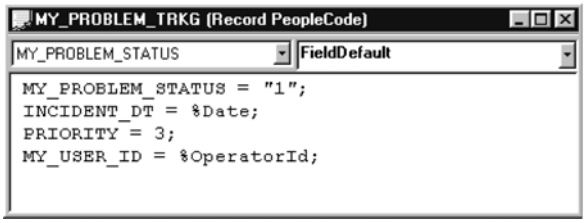


Figure 13.1
Example of FieldDefault PeopleCode

FieldFormula

Each time an event is triggered by the Application Processor, the FieldFormula event is also triggered. As a result, the PeopleCode contained in FieldFormula is executed in between the execution of other events. Any PeopleCode contained in the FieldFormula event of a panel field is executed regardless of the field in which the PeopleCode resides. Due to its high performance overhead and potential system

degradation, the use of FieldFormula is used primarily in function libraries where stored PeopleCode can be shared across many records and varying panel groups.

RowInit

RowInit PeopleCode can be used to control the initial appearance of a field or panel group control. The event is triggered each time the Application Processor encounters a row of data before the panel is displayed. RowInit PeopleCode is executed on all fields and rows in the panel buffer. The operator class/location contains two panels that use a common derived/work record. PeopleCode is inserted into the RowInit event of the derived/work record to display the number of rows in a scroll area. Because the code operates on two different panels and is contained in the same Record.Field, the code uses an If conditional statement (discussed in chapter 12). This code (figure 13.2) will work in Add as well as Update/Display modes.

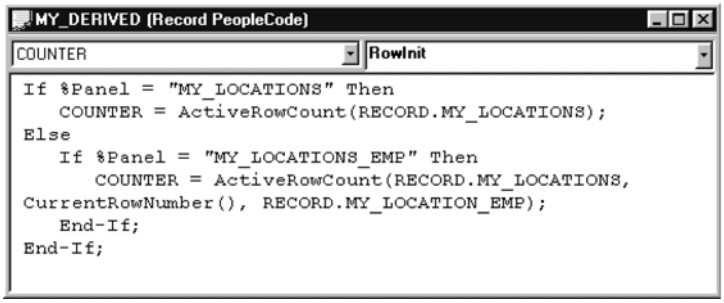


Figure 13.2 Shared RowInit PeopleCode for two panels

SearchInit

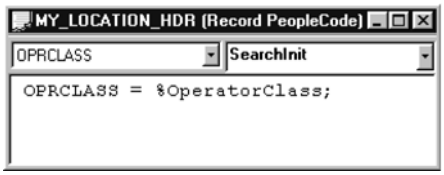


Figure 13.3 SearchInit PeopleCode

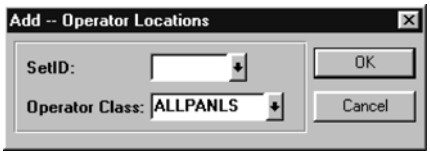


Figure 13.4 Add mode dialog box

In Add mode, the SearchInit event is triggered prior to display of the Add dialog. It enables the developer to prepopulate search fields. Under certain circumstances it may be necessary to populate a search dialog based on the action mode. This can be accomplished using the %Mode system variable discussed in chapter 12.

In the operator class/location example, the %OperatorClass system variable can be used to prepopulate the input dialog box. The SearchInit code returns the primary operator class for the current operator. The Add dialog is populated with this value. However, the user still has the option of modifying the operator class before pressing OK. The SearchInit PeopleCode and subsequent Add dialog box are illustrated in figures 13.3 and 13.4 respectively.

TIP `SearchInit` PeopleCode is only executed for search key fields. Any `SearchInit` PeopleCode attached to a non-search key field is ignored.

In the operator class/locations, `SETID` is an identifier that is five characters in length and Operator Class is eight characters. For this application, `SETID` and Operator Class are basic components required during initialization of the panel and are used to kick off the related Application Processor events. The events executed following the display and subsequent data entry into an `Add` dialog box are as follows:

- `FieldEdit`
- `FieldChange`
- `SaveEdit`
- `SearchSave`

FieldEdit

One of the most commonly used PeopleCode events is `FieldEdit`. Typical `FieldEdit` code includes edits such as date verification, hide/unhide of fields, and message displays to the operator when necessary. Figure 13.5 is an example of `FieldEdit` PeopleCode (applied to Problem Tracking) which generates an error message when the incident close date (CLOSE_DT) contains a date value prior to the incident date (INCIDENT_DT). `FieldEdit` is triggered when the user edits a field then tabs out or clicks on another field, edits a field and, without tabbing out, attempts to save the panel, changes the condition of a radio button, or clicks on a command push button.

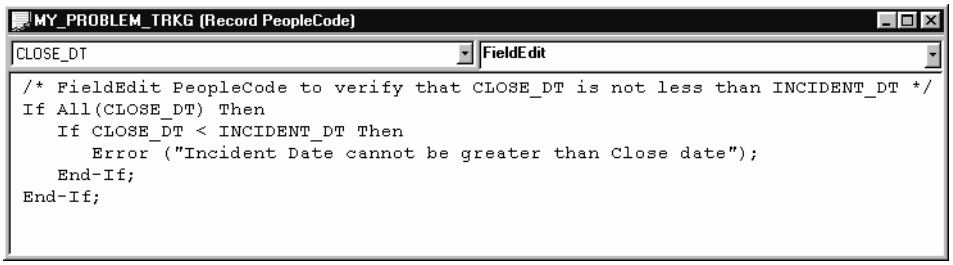


Figure 13.5 FieldEdit PeopleCode

When PeopleCode is added to validate fields, we can use the `Error` or `Warning` statement to display a message. Both statements highlight the field in red. The `Error` statement does not permit entering data into another field or attempting to save the changes after an error message has been displayed. The `Warning` statement notifies

the user with a message but permits work to continue. Any PeopleCode in FieldEdit is only triggered following Application Processor edits that include data type/format checking, prompt table verification, or Yes/No and translate table edits.

When the FieldEdit event is triggered from an Add dialog box, only the search fields that appear in the Add dialog box will have any associated FieldEdit PeopleCode executed. FieldEdit PeopleCode for non-search key fields which appear on a panel is executed after the panel is displayed and the FieldEdit actions take place.

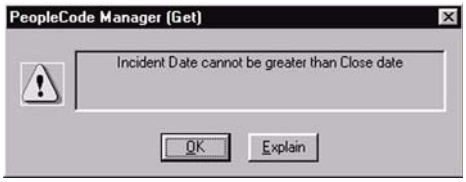


Figure 13.6 Error Message generated during FieldEdit

An error message similar to the one displayed in figure 13.6 prevents the panel or panel group from being saved. In the example, if an incident close date less than the incident date is entered, the FieldEdit event is triggered, regardless of whether or not we tab out of the field. When an error message is displayed, the choices available to the user are to correct the error or cancel the panel.

BEST PRACTICE

The example in figure 13.5 verifies that CLOSE_DT has a value before performing additional edits. This is accomplished using the All PeopleCode function.

FieldChange

The FieldChange event follows FieldEdit and occurs after the contents of a field have been modified and FieldEdit PeopleCode has been accepted. FieldChange is available to all modes (Add, Update/Display,Correction, Data Entry). As the name suggests, the event is triggered when a field on a panel is changed. After FieldEdit, the Application Processor then writes the changed field to the panel buffer and triggers the FieldChange event. PeopleCode in this event is commonly used to recalculate field values on a panel or to change the appearance of fields, buttons, or other controls on a panel.

SaveEdit

SaveEdit is another important event. After an Add dialog box is filled, and all edits have occurred for the search fields entered, any SaveEdit PeopleCode related only to the search fields is executed.

Because SaveEdit is used in all modes, an additional description will be presented in the section Data Entry or Inquiry.

SearchSave

PeopleCode in this event is executed for search key fields on a dialog (Add, Search and Data Entry dialogs). While `SearchInit` enables us to control processing before search keys are populated, `SearchSave` is used after the search keys are populated and the user has clicked OK. If we want to ensure that the user enters at least one value into a search field, we can do so with PeopleCode in this event. Figure 13.7 is an example of code used to require at least one key field. A partial key value is acceptable because it can be used to limit the number of rows returned in tables that contain large amounts of data.

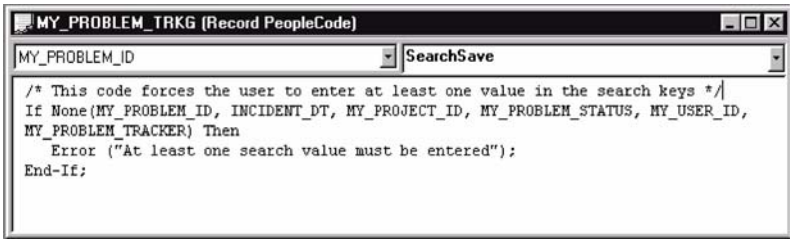


Figure 13.7 SearchSave code to require at least one search key

13.3 DATA RETRIEVAL

13.3.1 Search processing—Update mode

Choosing `Update/Display` from a menu generates a search dialog box. The search fields displayed are those defined as list box items in Application Designer. Before the Search Dialog box is displayed, the Application Processor triggers the following events:

- `SearchInit`
- `SearchSave`
- `RowSelect`

SearchInit

PeopleCode behaves the same regardless of mode. Therefore, it is important to know when and under what circumstances the code will be executed. Because `SearchInit` PeopleCode can be executed from `Add` as well as `Update/Display`, a common practice is to reference the `%Mode` system variable. A search dialog for the operator class/location panels is illustrated by figure 13.8.

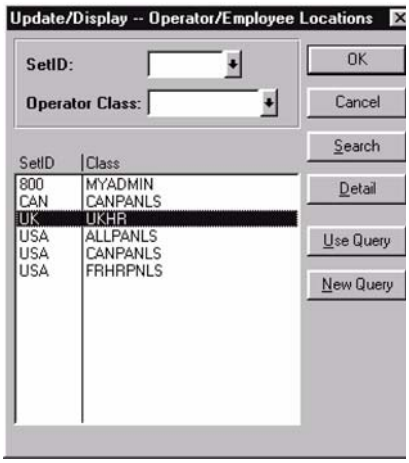


Figure 13.8 Search Dialog—Update mode

SearchSave

`SearchSave` is used after the search keys are populated and the user has clicked OK. The event behaves the same as in `Add` mode following the display and subsequent data entry into a dialog box.

RowSelect

This event occurs at the start of the Panel Build process when data are read into the panel group. `RowSelect` PeopleCode can be used to prevent the Application Processor from loading data, based on specific criteria. PeopleCode can also be used to stop the Application Processor from loading further rows of data. Without `RowSelect` PeopleCode, the Application Processor inserts data into the panel group when the database record key matches the designated search key.

How do we stop at a certain row or prevent more rows from being loaded once a certain condition has been met? In earlier releases of PeopleCode, the `Warning` and `Error` statements could have been used. Placing these statements in the `RowSelect` event has the effect of rejecting the row before it is displayed. Although not recommended, these statements can still be used. We can now, however, code a `DiscardRow` or a `StopFetching` function statement to achieve these tasks. The `DiscardRow` skips the current row of data and processing continues with any subsequent rows. Using the `StopFetching` function enables the Application Processor to select the current row of data and terminate reading any subsequent rows.

Here's a question: Would you code a `RowSelect` event with the knowledge that users may be loading many rows onto their clients? And, more importantly, what if the client is on the other side of the planet from the database server? Take note: `DiscardRow` or `StopFetching` discards rows or stops the selection process after rows have been accepted by the Application Processor. As a result, some overhead exists, as does the potential for inefficient coding with the implementation of `RowSelect` using these two functions. A small application may work well with these statements. However, a heavily used application would fare better with record views and effective dated tables where historical data could be separated before getting to the client.

In our application linking operator classes to office locations, we need to remove any employees who have an `EMPLID` starting with `'L'`. This process can be accomplished by inserting the `DiscardRow` function into the `RowSelect` event (figure 13.9). Because the operator class/location panel is not a heavily used

application, the concern over inefficient code is not as important as in some heavily used applications.

During the panel group build phase, `Update` action modes incorporate some events used during `Add`. The Application Processor can either add data to the panel group or retrieve data from database tables. Based on panel display control fields and related display fields, the Application Processor determines when to retrieve data from other records. As discussed in chapter 9, when building panels, the records can be SQL tables, SQL views, or derived/work records.

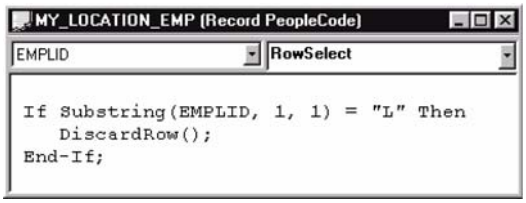


Figure 13.9
Using DiscardRow in RowSelect

During the data retrieval stage, there is little PeopleCode involvement. The Application Processor constructs SQL statements based on search key values provided previously, records defined as related display, and, more importantly, the mode in which the Application Processor is operating. During `Add` mode, less data are retrieved from the database than in modes such as `Update/Display` and `Correction`. After the Application Processor has retrieved the data necessary, it is ready to go into the panel group display stage.

13.4 PANEL GROUP DISPLAY

During Panel Group display, the following PeopleCode events are executed:

- `FieldDefault` (Iterative)
- `RowInit`

The previous section, "Search processing—`Add` mode" (13.2.2) covers `FieldDefault` and `FieldFormula` events. These events are categoried as Default processing. "Search processing—`Add` mode" also discusses `RowInit` in depth.

At this juncture, the Application Processor has selected (and possibly removed) records using `DiscardRow` in the `RowSelect`. The `RowInit` event has also occurred for the selected rows. The panel is now ready to be presented to the user. After the panel group is displayed, the Application Processor exits from the freeway and into a rest area where it waits for operator actions.

NOTE When selecting data in a scroll area, `RowSelect` and `RowInit` PeopleCode events occur for each row that is read into the scroll.

TIP All default processing and display processing PeopleCode events from panel fields, except related display fields, are processed during panel display. Even if a field is not present in the panel, PeopleCode events are triggered from that field. One exception to this rule is fields from Derived/Work records. PeopleCode is processed from derived fields only when they are present in the panel. For this reason the same Derived/Work record can be shared across multiple panel sessions without interfering with each other.

13.5 DATA ENTRY AND INQUIRY

13.5.1 Modifying data on a panel

From a PeopleCode perspective, the majority of events and activities take place during data entry and save processing. Save processing events begin when the user clicks the save toolbar icon or selects File, Save from the menu. When the panel group is displayed the user is presented with information based on his/her security profile. If the user has view capability only, then no adding or updating can be performed. Given the proper security, however, the user can perform activities that trigger PeopleCode events. These events include:

- `FieldEdit`
- `FieldChange`
- `RowInsert`
- `RowDelete`

NOTE When entering data on a panel, the F4 and CTRL+F4 prompt does not trigger any PeopleCode events.

FieldEdit and FieldChange

During data entry, when the user edits a field on the panel and then tabs out of the field or clicks on another field, the `FieldEdit` PeopleCode event is triggered. Please refer to section 13.2.2 for more information regarding `FieldEdit` and `FieldChange`.

RowInsert

When a `RowInsert` activity is performed (F7), the Application Processor adds a row of data to the active scroll area. In Problem Tracking, the panel MY_PROBLEM_TRKG is not a scroll panel and, therefore, PeopleCode events such as `RowInsert` and `RowDelete` are not executed. This is important because we could add lines and lines of `RowInsert` PeopleCode containing the fanciest routines,

functions and edits, but, if our screen does not contain a scroll bar, then our PeopleCode will never get executed!

The Application Processor first inserts an empty row into the active scroll area. The RowInsert event is then executed, and PeopleCode operates on the fields in the newly inserted row. A successful RowInsert event is followed by the Application Processor events FieldDefault, FieldFormula, and RowInit. RowInsert PeopleCode operates on rows containing a scroll area on the panel. These events are executed before the panel is redisplayed so if RowInsert code is replicated in the RowInit event, the code is triggered twice because RowInit follows RowInsert. RowInsert PeopleCode can be used primarily for specific processing when new rows are inserted into a scroll area. RowInit PeopleCode is used when rows of data are loaded into a panel buffer. RowInit is triggered for both scroll and nonscroll data, whereas RowInsert is only triggered for panels containing scroll data.

The Error and Warning statements should not be used in RowInsert PeopleCode. The Application Processor runtime error will force the user to cancel the panel group without saving it.

RowDelete

RowDelete results in removal of data from a scroll area and, subsequently, from the database during save operations. This event is triggered when the user deletes a row of data by pressing the row delete button or the F8 function key. Before the Application Processor removes a row from the buffer, fields on the deleted row can be referenced from PeopleCode in the RowDelete event. The Application Processor marks the rows as deleted, but the actual delete does not take place until save processing. The RowDelete event is executed following the "Delete current row?" verification message issued by the Application Processor. When a No reply is given, any code associated with RowDelete is not triggered. An application of the use of RowDelete can be used to update the Scroll Count field on the panel illustrated in figure 13.10. Figure 13.11 represents a simple line of code which updates the scroll count during a RowDelete event. The ActiveRowCount function is used to produce a count of the number of active rows in the panel scroll area. Rows marked for deletion are not included in the count.

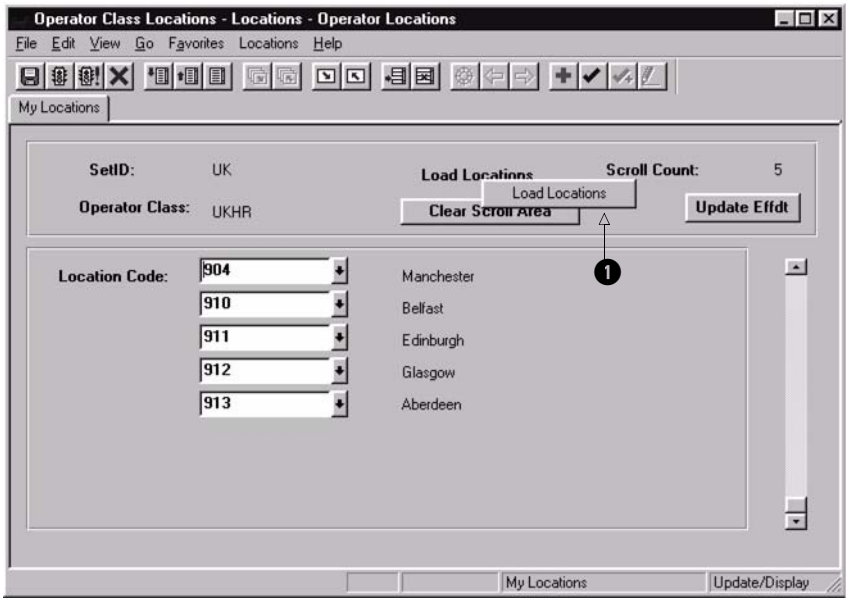


Figure 13.10 RowInsert and RowDelete processing is only possible on panels containing scrolls.

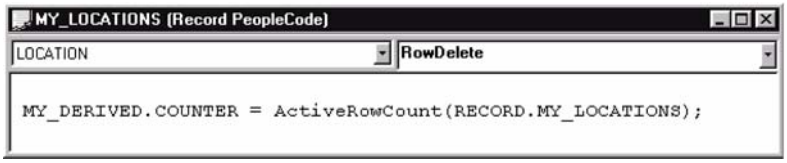


Figure 13.11 RowDelete code used to update counter

PrePopup

As developers, we can attach a pop-up menu to a panel field, which is displayed at runtime when the user clicks the right mouse button of the corresponding field. The `PrePopup` event is triggered immediately before the display of the pop-up menu. When we compare PeopleCode in pop-up menus to PeopleCode in standard menus, we see limitations to the standard menu PeopleCode that do not exist in pop-up menu PeopleCode. If we do not wish to add command push buttons to our panel, pop-up menus can be used instead. As an example, let's refer to figure 13.10 and, more specifically, the label titled "Load Locations," ❶. This field is associated with a pop-up menu which loads locations associated with the SETID field. For this process we have defined a pop-up menu named LOAD_LOCATIONS. To illustrate the `PrePopup` event, let's assume that after we right-click the field labeled "Load Locations," the

PeopleCode associated with this pop-up menu is executed (figure 13.12). Each time the right mouse button is clicked, the scroll area fills with data. An unsuspecting end-user can perform this several times before activating the save button. This type action generates an error message indicating duplicate keys. We can be pro-active and prevent this duplicate key situation by including code to delete data before populating the scroll buffer area. In this example, the code to clear the scroll area and related database records is inserted into the `PrePopup` event (figure 13.13). As a result of implementing the `PrePopup` code, when the user right-clicks the "Load Locations" field, the `PrePopup` event executes the associated code and clears the scroll area before it is filled with data.

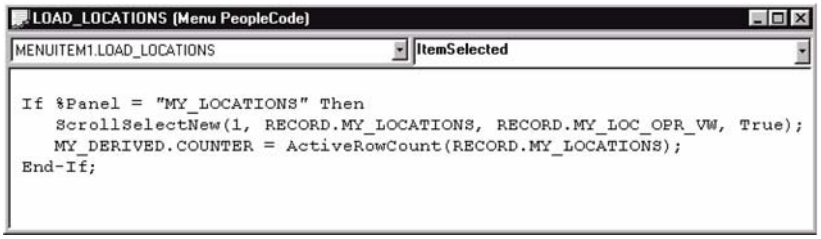


Figure 13.12 PeopleCode associated with a pop-up menu

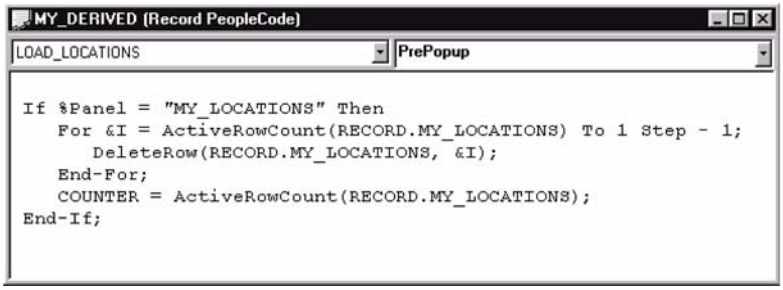


Figure 13.13 PrePopup code to clear scroll buffer area

Pop-up menu PeopleCode could also be used to calculate field values, trigger `Workflow` events, or run a modal transfer.

NOTE The `DeleteRow` function used in figure 13.13 could have been combined with the code in figure 12.12. It has been placed into the event to demonstrate `PrePopup`.

13.6 SAVE PROCESSING

After data entry is completed and all FieldEdit PeopleCode events are executed, the user can either save the panel or cancel it. As a result of canceling a panel, no save processing is performed. All those elegant routines written for execution during a save process are overlooked by the Application Processor, which resets the status of the panel group following a cancel. When the save button is activated, the Application Processor then triggers four events associated with save. The events are:

- SaveEdit
- SavePreChg
- Workflow
- SavePostChg

SaveEdit

The PeopleCode associated with this event is executed after the operator saves the panel group. This event allows the developer to verify data consistency across fields and records.

After the user has clicked the save button or pressed enter, the SaveEdit event is triggered. Any PeopleCode in this event is executed after the Application Processor performs edits of its own. SaveEdit PeopleCode can be used when more than one field in the panel group is required to perform validity checking or a consistency edit. The SaveEdit event enables us to verify fields after they have all been keyed into a panel group. We could use FieldEdit PeopleCode, but if there are ten fields on a panel group that require some form of editing, a user would not want to enter a field, have a message appear, and correct the field only to repeat the same sequence on the next field. SaveEdit is not related to any specific field and, except for deleted rows, is triggered on every row of data in the panel group buffer. This can be tricky at times. As an example, let's assume that we have an employee's JOB history panel, which is effective-dated. Let's also assume that we've made changes to the current effective-dated row. Coincidentally, a record somewhere in the scroll contains bad data. The data are probably the result of a conversion program that did not perform edits on every field. When the user clicks "Save," the associated PeopleCode will be triggered for every row in the scroll. Any errors encountered somewhere in the stack of JOB data will require correction when the error prevents a save operation.

Our friends, the Error and Warning statements, can be used in SaveEdit PeopleCode. The Error statement displays a message, but prevents the user from saving any data while the Warning statement displays a message and presents the user with the option to save the data via an OK or to Cancel without saving. If the PeopleCode program can determine which field requires correction, the PeopleCode function SetCursorPos can be called before Error or Warning. Figure 13.14 illustrates PeopleCode in the SaveEdit event. The code verifies for changed fields using the built-in function FieldChanged and calls an External PeopleCode function named

MyAuditFunction. Putting this code into an event such as FieldEdit or FieldChange represents a bit more of a challenge because the function would be called each time the event is generated. Think of the user who changes a field many times before pressing save.

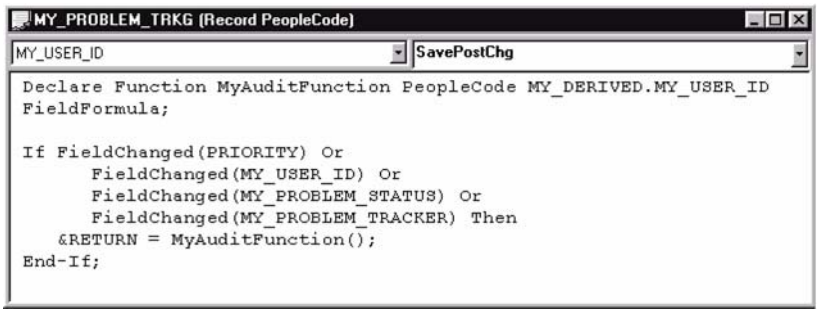


Figure 13.14 SaveEdit PeopleCode

SavePreChg

SavePreChg occurs after the SaveEdit event terminates successfully. SavePreChg enables the PeopleCode program to modify data one last time before the Application Processor updates the database. Upon successful completion of a SavePreChg event, the WorkFlow event is then executed. WorkFlow is triggered before the Application Processor generates the corresponding database updates, such as INSERT, UPDATE, and DELETE.

WorkFlow

This event follows SavePreChg and is executed before any of the database updates are performed. The database updates are then followed by the SavePostChg event. Workflow PeopleCode is separated from PeopleCode in the other events and should be designed so that Workflow executes when all SaveEdit and SavePreChg code have been completed. Workflow programs should include PeopleCode related to Workflow and linked to business processes only. At the time a business process is defined, we also include the panels that trigger any related business events. The combination workflow panels and PeopleCode programs are considered application agents and are loaded as part of the panel group.

Workflow PeopleCode includes either the TriggerBusinessEvent() a PeopleCode function that triggers events, or Virtual_Router(), which is linked to the Virtual Approver.

TIP The code should verify when the business event is to be triggered; this is referred to as a business rule.

SavePostChg

After `SavePreChg`, the Application Processor executes PeopleCode in the `Workflow` event. The Application Processor then executes any database updates (`INSERT`, `UPDATE`, `DELETE`), followed by the `SavePostChg` event. Records that require updating, but are not included in the current panel group, can be updated in this event using the `SQLExec` statement. Be aware that if the `SavePostChg` event fails to execute correctly, the Application Processor will not issue a SQL commit. Similarly, the use of `Error` and `Warning` statements in `SavePostChg` PeopleCode issues a runtime error that forces the user to cancel the panel group without saving it.

NOTE It is important to know that an unsuccessful `Workflow` event prevents the `SavePostChg` event from executing.

13.6.1 Adding PeopleCode to save processing

Some code added to the Problem Tracking application makes it more efficient, but there is always room for improvement. It is not always a good idea to add "bells and whistles" to an application just because we have the tools. It is also not feasible to add complex code to satisfy one user when there are hundreds of other users that will not benefit from the added code. The application presented is a small application, and some "loop holes" are opened up, as we will illustrate.

When a problem has been resolved, certain housekeeping steps should be completed. These steps include entering the Problem Resolution text and entering a Close date.

In its current form, the application will accept a problem status value of 5 (Resolved) without validating that the resolution text field is entered. Because we cannot change the Record Field Properties of the MY_PROBLEM_RESOLTN field and make it required, PeopleCode can be added to accomplish this task for us. The code shown (figure 13.15) will accomplish the task and require that the resolution text field is entered for resolved issues.

The next step is to ensure that the Close Date field is entered when the problem status has been resolved. The code in figure 13.16 will check for a status of 5 and then require that the CLOSE_DT field contain a valid date value. Because CLOSE_DT is defined as a `Date` data type, the Application Processor performs the standard date checking for us.

One additional piece of code we would like to add is related to the following scenario. Let's say the problem is reported and entered into the system. The problem status is subsequently assigned, tested and resolved; and at the time of resolution, the

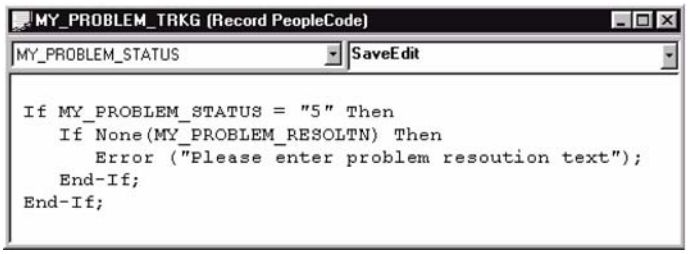


Figure 13.15 Using PeopleCode to verify a field is entered

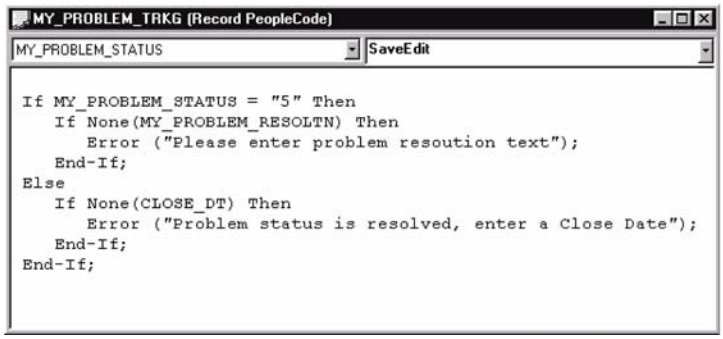


Figure 13.16 Verify that Close Date is entered

Problem Resolution text field is entered, along with a close date. It is then determined that the problem was not resolved and the status should be set to "In Progress," to indicate that it is being handled by a developer.

This issue requires that we reset the Problem Resolution text field and the Close date. Our program logic has now changed from reactive to proactive. The code to accomplish this task is illustrated in figure 13.17. The PeopleCode function `SetDefault` is used to set the field to a `NULL` value. During the next `FieldDefault` event, MY_PROBLEM_RESOLTN and CLOSE_DT are set to their appropriate default values. After the user has clicked the save button or pressed the ENTER key, the `SaveEdit` event is triggered. The last several examples were placed into the `SaveEdit` event of the field MY_PROBLEM_STATUS. The code is placed into `SaveEdit` to ensure that is it executed after the Application Processor performs its own edits. Additionally, `SaveEdit` PeopleCode is used when more than one field in the panel group is required to perform validity checking. Another less strategic reason to include the code in `SaveEdit` is that some end-users will automatically remove the resolution text field and close date then reset the problem status to "2." This action will bypass the edit in figure 13.17.

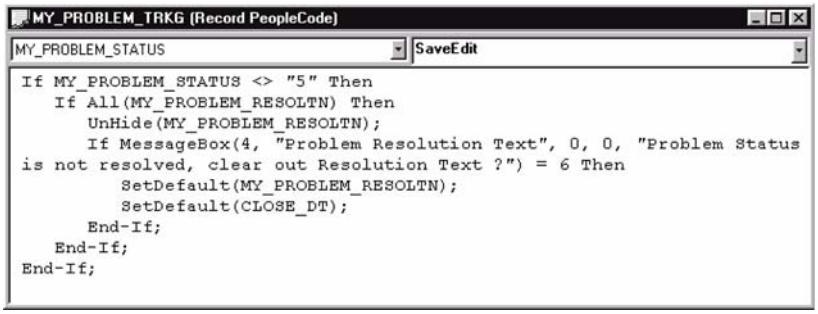


Figure 13.17 Code to reset data

Figure 13.18 illustrates the completed panel with some bells and whistles added for additional editing features.

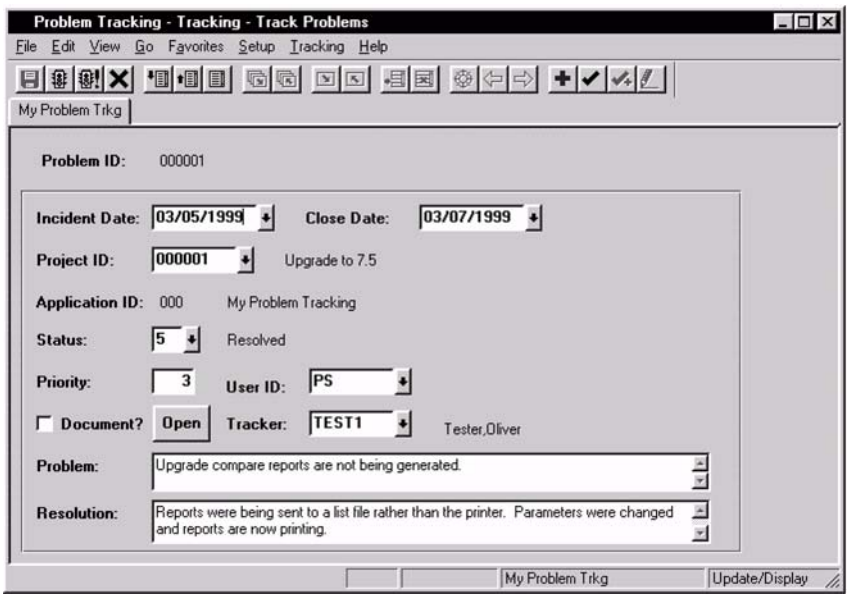


Figure 13.18 Completed panel with custom PeopleCode

KEY POINTS

1. The Application Processor acts as a traffic agent on a busy street. It controls everything from the time the user selects a menu item until all save processing has been completed as well as everything in between.
2. Event Processing is the cornerstone of PeopleCode programs. Code can be inserted before a search dialog is displayed, at every step in between, and after all database updates have been performed.
3. Some events such as `FieldDefault` and `RowInit` occur before data are displayed on the panel.
4. `FieldFormula` is an event that occurs after each and every event that contains PeopleCode is executed. A lot of overhead is associated with `FieldFormula` and its use has been left mostly to function libraries on Derived/Work records. `FieldEdit` also executes for each field on a panel which contains a blank or zero value.
5. When data are entered into a panel, events such as `FieldEdit` and `FieldChange` can be executed before save processing.
6. `RowInsert` and `RowDelete` are events linked to the F7 and F8 function keys, respectively. They are used primarily on panels containing scroll bars.
7. After the save button is pressed, events such as `SaveEdit`, `SavePreChg`, and `SavePostChg` are executed.
8. The `WorkFlow` event follows `SavePreChg` just prior to all database table updates.
9. Any `PrePopup` code associated with a pop-up menu is executed before the pop-up menu is displayed. If we do not wish to add command push buttons to a panel, pop-up menus and any associated PeopleCode can always be used instead.

CHAPTER 14

Messages and error handling

Communication between a PeopleSoft application and an end-user is key to a successful implementation. PeopleCode message functions enable the application to send messages containing a simple OK push button or an Error message that will prevent further processing of a panel, until data on the panel has been corrected. In between these extremes, messages can be sent in varying detail containing custom messages. Utilizing the PeopleTools Message Catalog, these messages can be sent to a worldwide audience in a variety of languages. With no changes to a PeopleCode program and a change to the language code, cataloged messages can be "cloned" for every language required.

14.1 USING THE MESSAGEBOX FUNCTION

The `MessageBox` function is one of the more important PeopleCode built-in functions. `MessageBox` enables PeopleCode programs to communicate with an end-user via a message box window. PeopleCode program flow can then be controlled based on the push button return values selected. (The syntax and rules of `MessageBox` can be found in appendix E.)

One of the parameters used by `MessageBox` is style. The style parameter enables the construction of a message box window with a blend of icons and push buttons. An optional default button can also be specified. Table 14.1 identifies the values required for buttons, default buttons, and icons.

Table 14.1 Style parameter combinations

Value	Push buttons	Default button	Type of icon
0	One push button containing OK	First button in the message box is the default	None
1	Two buttons containing OK and Cancel		
2	Three buttons containing Abort, Retry and Ignore		
3	Three buttons containing Yes, No and Cancel		
4	Two buttons containing Yes and No		
5	Two buttons containing Retry and Cancel		
16			Icon contains a stop sign. 
32			Icon contains a question mark. 
48			Icon contains an exclamation point. 
64			Contains a small letter "i." 
256		Second button in the message box is the default.	
512		Third button in the message box is the default.	

In addition to the button combinations illustrated in table 14.1, the MessageBox window also contains an Explain button. As the name implies, the Explain button provides a more detailed explanation of the message text.

Some important rules regarding the use of MessageBox concern the buttons that can be used during certain PeopleCode events. The OK and Explain buttons do not interrupt processing. However, when any other type of button displayed in a message box interrupts processing, the system waits until one of the buttons is clicked. The function then becomes "user think-time," which indicates the button action returns a value to the function. As a result of awaiting a reply, the Application Processor suspends the PeopleCode program until the user clicks on one of the buttons contained in the message. A program in this manner cannot be used in the following events:

- RowSelect
- SavePreChg
- WorkFlow
- SavePostChg

When the OK button is the only button represented by the style parameter, the function can then be used in all PeopleCode events. The style parameter does not control the Explain button and is not indicative of whether MessageBox is considered think-time.

Based on table 14.1, if a message box contains an OK and Cancel button as the default, along with a question mark icon, what would the value of style have to be? If you guessed 289, you win the prize! The following PeopleCode statement produces the message box shown in figure 14.1.

```
MessageBox (289,"Example", 0, 0, "MessageBox example");
```

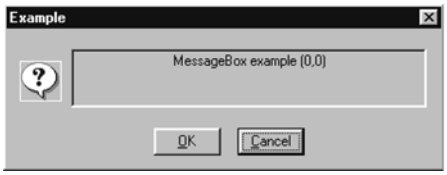


Figure 14.1 A standard MessageBox window

Using table 14.1 as a reference, let's examine the MessageBox window in figure 14.1. The MessageBox contains two buttons, OK and Cancel (style value = 1). The second button represents the default (style value = 256), and the question mark icon appears in the message box (style value = 32). The sum of these values is 289, which is the style used for this example.

Let's assume the numbers were not added correctly and, rather than using 289, we supplied 288 as the style to MessageBox instead. How will the message box appear? When the following statement is executed, the message box appears as shown in figure 14.2.

```
MessageBox (288,"Example", 0, 0, "MessageBox example");
```

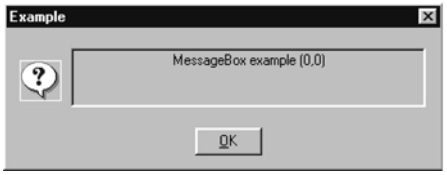


Figure 14.2 MessageBox window using erroneous style

Because the style parameter is incorrect by a value of 1, the message box contains an OK push button only. The message box contains this single push button because zero is the value used to represent a single button containing OK (as described in table 14.1). A value of 288 indicates that any additional buttons are not factored into the required style value equation.

The next parameter is the title of the message box. The title is defaulted when a null string is passed. For example, the following statement produces a null title in the `MessageBox` window:

```
MessageBox (289,"", 0, 0, "MessageBox example");
```

The statement produces the message box shown in figure 14.3 with a defaulted title.

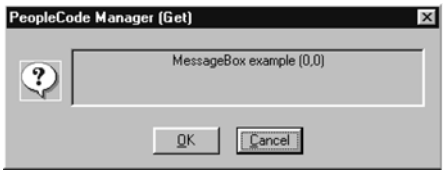


Figure 14.3 MessageBox with defaulted title

The next parameter is the message_set number of the message catalog. The Message Catalog enables us to store messages for retrieval using functions such as `MessageBox`. We can either set up our own custom messages or use the existing ones. To define a custom cataloged message, we can select Use →Message Catalog from the Utilities menu.

Navigation: Utilities →Use →Message Catalog →Add

Figure 14.4 Adding Message Catalog entries

The illustration in figure 14.4 indicates message set number 20001 in English will be added to the catalog. Message sets 1 through 19,999 are reserved for PeopleSoft application use. Message sets 20,0000 through 29,000 are available for custom use.

We populate the panel in figure 14.5 and then proceed to add the same message in Spanish, as illustrated by figure 14.6.

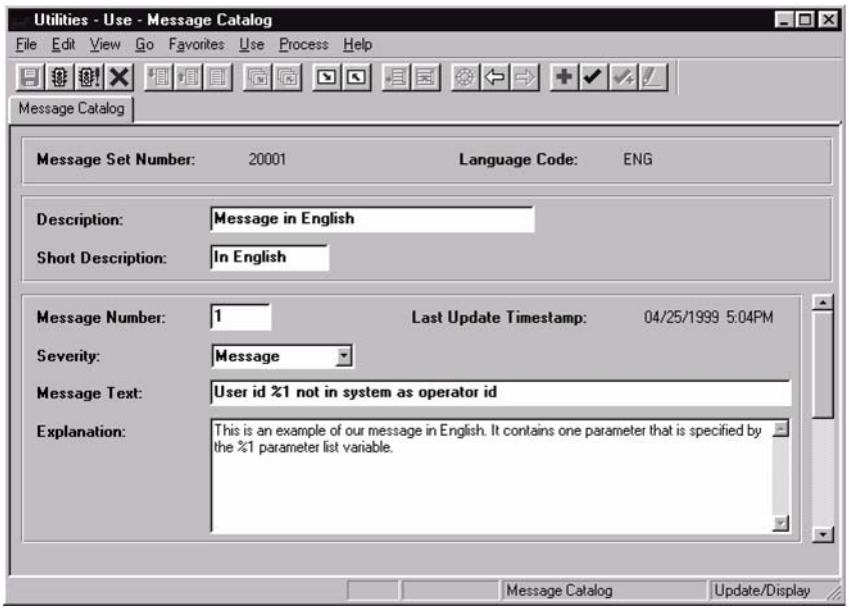


Figure 14.5 Message in English

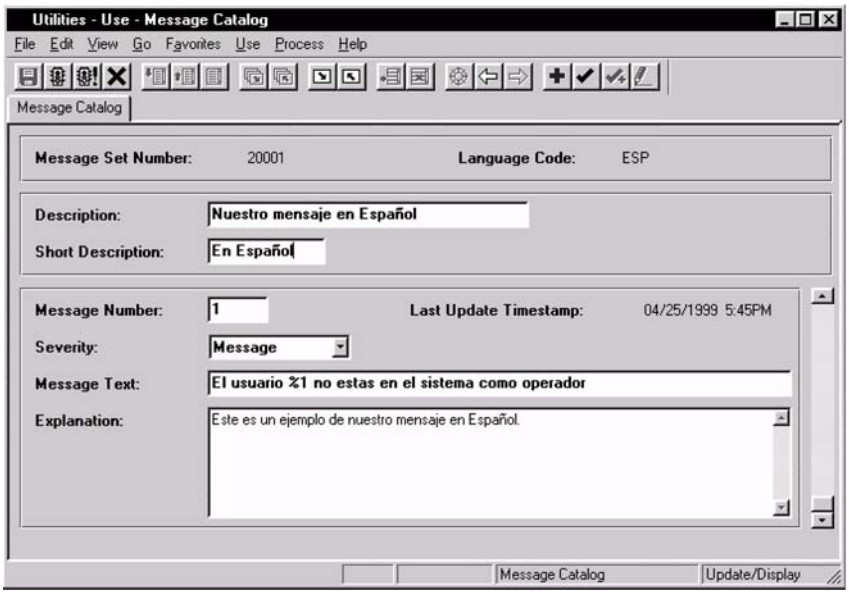


Figure 14.6 Same message number in Spanish

NOTE Messages added to the range used by PeopleSoft (1–19,999) or messages customized in this range may be impacted by future releases of the product.

The next `MessageBox` parameter is the message number. Each message set is accompanied by one or more messages identified by the message number. The examples in figures 14.5 and 14.6 begin with message number 1 (a most logical starting place). The language, description, and short description fields are part of the message set. The message number is automatically assigned to the next available number in the message set and includes the severity, message text, and explanation. To insert an additional message number for the message set 20001, we first display the message displayed in figure 14.5. The next step is to use the Insert Row toolbar icon or F7 function key. The Insert Row augments the message number and presents a panel that enables the entry of message severity, text, and explanation. A similar panel is shown in figure 14.7.

Default text is the next parameter and it is displayed under two circumstances. The first occurs when the message identified by message set and message number is not found in the message catalog. The second condition occurs when a value less than 1 is supplied in the message set field. A `MessageBox` as outlined in the following statement uses the default text because of the zero passed in the message set number.

```
MessageBox(289, "", 0, 0, "");
```

Figure 14.7 Inserting additional message number

Figure 14.8 Missing default text

The preceding example specifies a null default text. As a result, the Application Processor issues the message shown in figure 14.8.

As illustrated by figure 14.8, `MessageBox` uses the style parameter and replaces the title with a default value. `MessageBox` then fills in the default text with its own `MESSAGE NOT FOUND` verb.

The next and last parameter is the parameter list. It may sound somewhat redundant, but these are the optional parameters that appear in the message or default text. Variables cannot simply be imbedded into messages, they must be passed as parameters to the message text. The parameters are referenced using the `'%'` character and an integer that corresponds to the sequence in which the parameter appears in the message text. To include the literal `'%'` in the text use `'%%'`.

MessageBox return value

`MessageBox` returns a value based on which push button is pressed. The return values are listed in table 14.2. A value of zero returned by `MessageBox` indicates that there is insufficient memory to create the message box. A message box containing a Cancel button returns the same value as when the ESC key is pressed or Cancel button is selected

Table 14.2 MessageBox return values

Returns	Description
0	Insufficient memory
-1	Warning
1	OK button was pressed
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Message severity

When used with cataloged messages, `MessageBox` can specify a message severity. Message severity is a parameter entered into the message catalog. By specifying the message severity, a message can be transformed from a warning into an error message without a change to the PeopleCode that generates the message. To modify the message severity for a cataloged message, use the Utilities menu.

The illustration in figure 14.9 indicates that there are four message severity categories and outcomes:

- *Message* Output is displayed as a message only.
- *Warning* Message text is sent as a warning and requires user response.
- *Error* The message is displayed with an error, which implies that further processing is halted until the error is corrected.
- *Cancel* After the message is displayed, cancellation of the current panel is forced.

Navigation: Utilities →Use →Message Catalog →Update/Display

Figure 14.9 Message severity levels

Utilizing MessageBox in PeopleCode

The following example performs some verification and generates the message that was added in figure 14.5. The `MessageBox` portion of the code contains a style of 289. As we learned, a style of 289 includes an OK and default Cancel button in addition to a question mark icon. When the message set number is specified and is not less than 1, the Explain button is also displayed. This occurs even when the message does not appear in the message catalog. When a non-zero message set number is supplied, the message stored in the catalog table is displayed in the message window. However, if the message is not stored in the catalog, the default text is then substituted. The message

text parameter list contains one parameter identified by `'%1'`. When the message is displayed, `'%1'` contains the value of the field MY_USER_ID. This PeopleCode example is utilized when setting up users in the current Problem Tracking application.

The code surrounding `MessageBox` verifies the existence of the user ID field on the PSOPERDEFN record. The panel field MY_USER_ID is compared to the contents of the system variable `%OperatorId`, which contains the uppercase string of the operator ID logged on. Assuming the `%OperatorId` variable contains the value `PS`, the following code generates the message displayed in figure 14.10 when executed on the panel:

```
If MY_USER_ID <> %OperatorId Then
   If (SQLExec("Select  'x' from PSOPRDEFN where OPRID = :1", MY_USER_ID,
&OPRID)) = True Then
      If %SqlRows = 0 Then
         MessageBox(289, "Verify", 20001, 1, "User id %1 not in system as
operator id", MY_USER_ID);
      End-If;
   End-If;
End-If;
```

Using the example in figure 14.10, under a similar circumstance a Spanish language user in Latin America who has a Spanish language setting receives the message shown

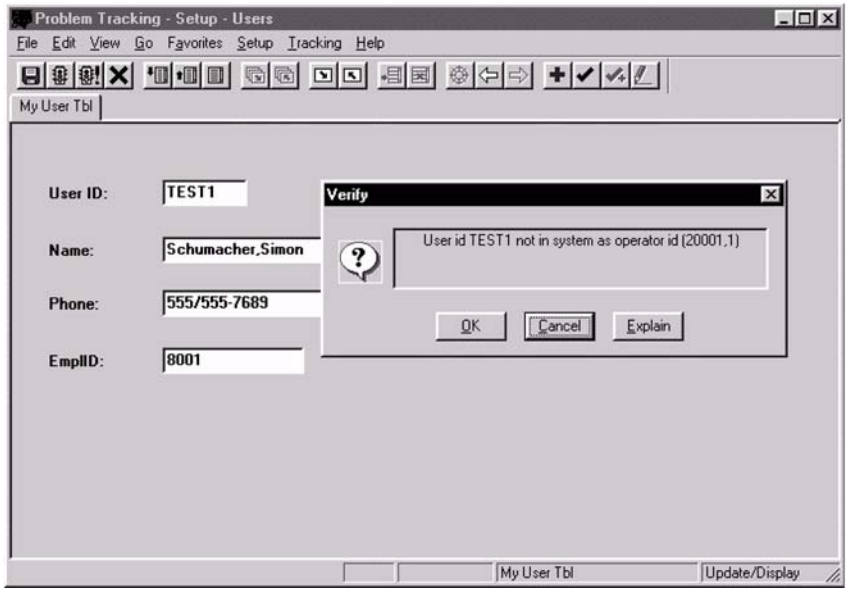


Figure 14.10 Our MessageBox in action

in figure 14.11. The beauty of this is that no additional changes are made to the PeopleCode program.

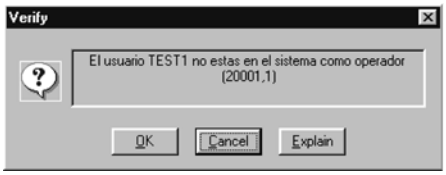


Figure 14.11 Display cataloged Spanish message

The message in figure 14.11 is message set 20001, message number 1. The Spanish language message text reflects the language code ESP. When a message number is duplicated in varying languages, all the other parameters including push buttons, defaults, and the question mark icon are identical to the original language message. The possibility exists that when a message is translated from one language to another, the entire message can be different. The example in figure 14.11 could relay a completely different message than the original intended message in figure 14.10. For implementations with a worldwide audience, some type of audit should be in place when messages are entered in different languages.

User Think-Time

At the beginning of this section, we discussed the incidence of User Think-Time. This can occur during specific PeopleCode events and can be characterized by a message box containing buttons other than OK and Explains. These "other" buttons interrupt system processing and wait until some action is performed with reference to the buttons. Because the system is in a wait mode, Application Processor suspends the PeopleCode program. `SavePostChg` is one event that does not allow user think-time functions.

Figure 14.12 is an example of a User Think-Time function.

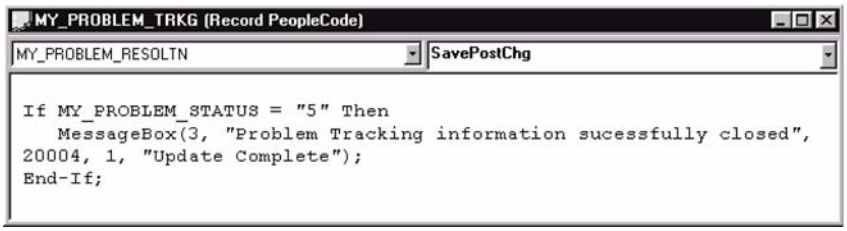
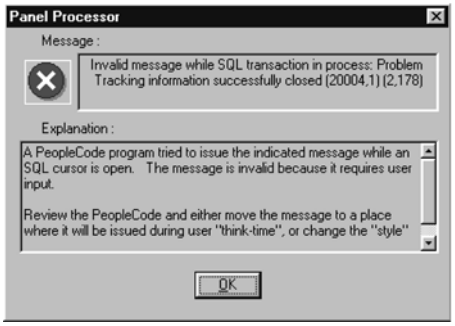


Figure 14.12 User Think-Time function

This is the type of code which does not get noticed immediately, but lurks until a Problem Tracking incident is resolved. When a problem has finally been resolved and all the information in the resolution text is entered, a message is displayed following save action. The message, shown in figure 14.13 is not the message expected by the user.



This is not the type of message that the developer who implemented the `SavePostChg` PeopleCode had in mind when it was coded. The error occurs because the style parameter contains three buttons: Yes, No, and Cancel. The PeopleCode can be corrected as follows:

Figure 14.13 User Think-Time message

```
If MY_PROBLEM_STATUS = "5" Then
   MessageBox(0, "Problem Tracking information sucessfully closed", 20004,
1, "Update Complete");
End-If;
```

The message box can now be displayed (figure 14.14). Because this is a problem tracking application, the users were kind enough to enter this into the system as well.

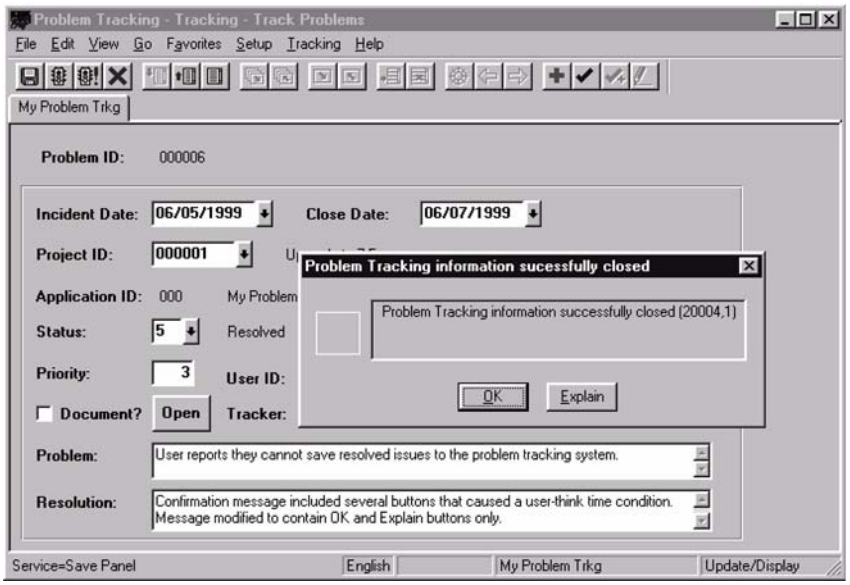


Figure 14.14 Successful display of message during SavePostChg

14.2 USING WINMESSAGE

Another common method of communicating messages is through the use of the WinMessage function. WinMessage can be used to send messages in a manner similar to MessageBox. As we will see in a later chapter, WinMessage can also be used as a debugging tool.

14.2.1 WinMessage

WinMessage is used to display an informational message box. With the use of the style parameter, two or more buttons can be included in the message, but their use is limited to certain PeopleCode events. When the style parameter is left out of the function call or contains more than one button, the function becomes user think-time, which indicates the button action returns a value to the function. As a result of awaiting a reply, the Application Processor suspends the PeopleCode program until the user clicks on one of the buttons contained in the message. A program in this manner cannot be used in the following PeopleCode events:

- SavePreChg
- WorkFlow
- RowSelect
- SavePostChg

From a debugging perspective, WinMessage can also be used to display field contents while allowing us to "inch" our way through PeopleCode when necessary.

A simple WinMessage can be written as follows:

```
WinMessage ("This is a basic WinMessage example");
```

Example 1

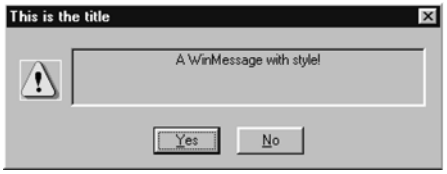


Figure 14.15 Message generated by WinMessage

WinMessage can also accept a style parameter similar to the style used in MessageBox. The parameter is optional and, when used, the number of push buttons, default button and type of icon that appear in the message box can be controlled based on the value passed. (Refer to table 14.1 for a list of style categories and values.) A third parameter, which contains a message box title, can also be supplied. The following code generates the message shown in figure 14.15:

```
WinMessage("A WinMessage with style!", 52, "This is the title");
```

Example 2

Let's assume we are examining a PeopleCode program in HRMS and want to know the values of three fields. The fields are COMPRATE, COMP_FREQUENCY, and STD_HRS. `WinMessage` can then be used to concatenate and display fields, variables, and literal strings. The PeopleCode utilizing `WinMessage` might be similar to the illustration in figure 14.16. The resulting message is in figure 14.17.

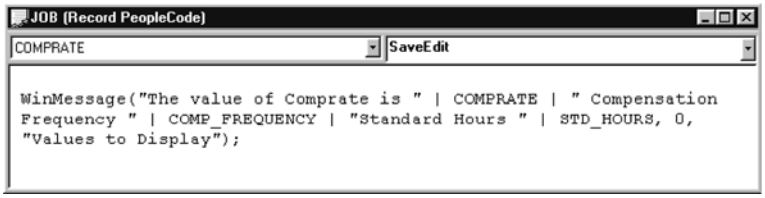


Figure 14.16 WinMessage used to display field values

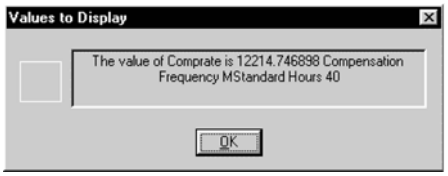


**Figure 14.17
Concatenated message using fields
and literal strings**

14.2.2 Additional examples

When fully utilized, the `WinMessage` function takes three parameters. The example in figure 14.16 cannot contain a comma in the message portion of the text string. Any record fields or variables displayed are concatenated with the message text. In the example, there are three text strings and three record fields. Initially, you may decide to code the statement differently. Here are some examples of what to expect:

If you separate the message text components with commas as illustrated by figure 14.18, expect the PeopleCode editor to return a message (figure 14.19) when performing a syntax check.

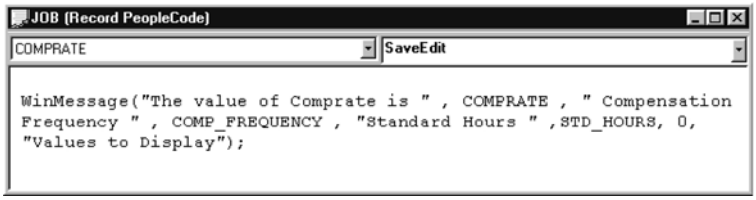


Figure 14.18 Incorrect use of commas in message parameter

Figure 14.19 Message returned by PeopleCode editor

You may decide to enclose the message text, record fields or variables within one string. If the code is written as outlined in figure 14.20, you will then be able to perform the syntax check correctly, save your PeopleCode, and move on to another task. However, when this is tested—or if you are the type of developer who moves work to production without testing (shame on you)—then expect a similar message to the one shown in figure 14.21.

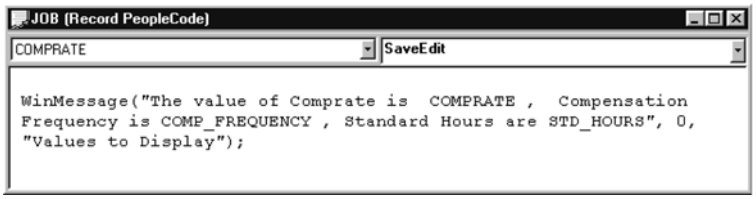


Figure 14.20 Invalid concatenation of fields and variables

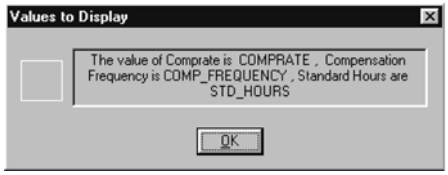


Figure 14.21 Incorrect message string

In the message displayed by figure 14.21, the Application Processor could not distinguish between record fields, variables, and text strings in the message parameter. This is why the concatenation character `'|'` is utilized to include strings and variables when used in such message.

TIP All data types are converted to `String` when used in `WinMessage`.

14.3 ERROR AND WARNING

When performing data validation it is sometimes necessary to stop a program and display a message. Among the various methods used to display messages, PeopleCode provides two functions that enable us to communicate with the end user. The `Error` and `Warning` functions can be used to perform these tasks.

14.3.1 Error

The `Error` function is used to display an error message and stop processing of the active panel. In a manner similar to `MessageBox`, `Error` works with messages stored in the Message Catalog or with a text string supplied to the Error function.

A basic `Error` statement can be written:

```
Error ("This is an example of an error message");
```

The value contained in String can be a literal text message or a message stored in the Message Catalog. A stored message must be retrieved using the MsgGet or MsgGetText function. This is important when using translated text messages as in figure 14.6. The Error function, when executed, terminates the PeopleCode program and prevents further statements from being executed. Error, however, produces varying results from one PeopleCode event to another. The events in which Error is usually incorporated include FieldEdit and SaveEdit. When executed in these events, the message is displayed and processing is halted. In FieldEdit, the field containing the PeopleCode event is highlighted. When used in SaveEdit, no fields are highlighted. One way we can work around this in the SaveEdit event is to use the SetCursorPos function for the field, prior to calling the Error function. RowDelete is another PeopleCode event in which Error is sometimes used. When Error is called from RowDelete, the message is displayed, and the row is not deleted.

The use of Error in other PeopleCode events is not recommended. These events include:

- FieldDefault
- FieldFormula
- RowInit
- FieldChange
- Prepopup
- RowInsert
- SavePreChg
- SavePostChg

The following illustrates the use of Error:

```
If MY_USER_ID <> %OperatorId Then
   If (SQLExec("Select  'x' from PSOPRDEFN where OPRID = :1", MY_USER_ID,
&OPRID)) = True Then
      If %SqlRows = 0 Then
         Error (MsgGet(20001, 1, "User id %1 not in system as operator id",
MY_USER_ID));
      End-If;
   End-If;
End-If;
```

14.3.2 Warning

The Warning function is used to display a warning type message. Warning differs from Error because processing is not halted by a warning message. The user is presented with OK and Explain buttons, then has the opportunity to correct or change data. Warning works with messages stored in the Message Catalog or a text string supplied to the Warning function.

A basic Warning statement can be written as:

```
Warning ("This is an example of a warning message");
```

The value contained in the string passed to `Warning` can be a literal text message or a message stored in the Message Catalog. The stored message must be retrieved using `MsgGet` or `MsgGetText`. `Warning` produces varying results from one PeopleCode event to another. The events in which `Warning` is commonly used include `FieldEdit` and `SaveEdit`. When used in `FieldEdit`, the message is displayed and the field that contains the PeopleCode is highlighted. Placing the `Warning` statement in `SaveEdit` displays the message but does not highlight fields. One way we can work around this in the `SaveEdit` event is to use the `SetCursorPos` function for the field prior to the `Warning` function call. `RowDelete` is another PeopleCode event in which `Warning` is sometimes used. When `Warning` is called in `RowDelete`, the message is displayed with OK and Cancel buttons. The user then has the option to either delete the row by pressing OK or to back out of the delete by pressing Cancel.

The use of `Warning` in other PeopleCode events is not recommended. These events include:

- `FieldDefault`
- `FieldFormula`
- `RowInit`
- `FieldChange`
- `RowInsert`
- `SavePreChg`
- `SavePostChg`

Figure 14.22 illustrates the use of the `Warning` statement in the Operator Class/Location panel. The resulting warning message text and subsequent explanation are shown in figures 14.23 and 14.24. The definition for message set 20001, message number 2 is shown in figure 14.25. `Warning` is called from the `RowDelete` event and contains an OK, a Cancel, and an Explain button.

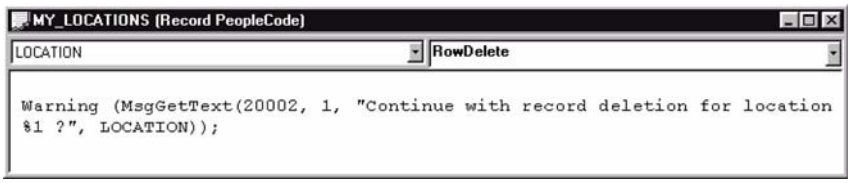


Figure 14.22 Using the Warning statement

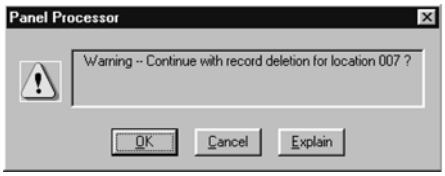


Figure 14.23 Warning message text

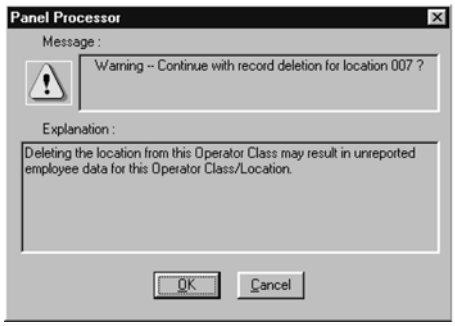


Figure 14.24 Explanation associated with Warning

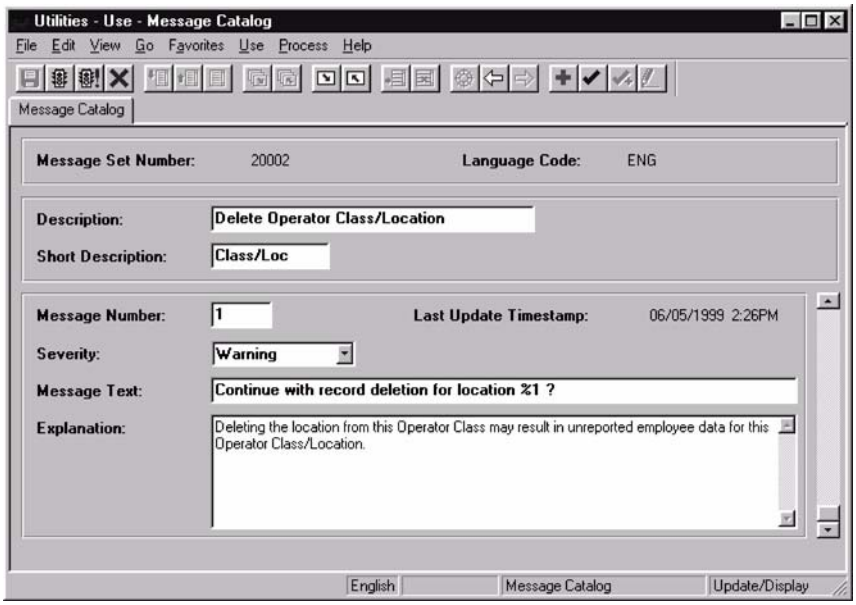


Figure 14.25 Message Number 2 definition

14.4 MSGGET AND MSGGETTEXT

Various PeopleCode message functions can be used to display messages that are stored in the Message Catalog table. Retrieving stored messages can be accomplished using the `MsgGet` and `MsgGetText` functions. As you will remember from an earlier discussion, the Message Catalog enables us to define the same message in various languages. These functions become an invaluable tool when working on a global application.

14.4.1 MsgGet

MsgGet has two primary tasks: to retrieve messages from the Message Catalog; and to substitute the value of each parameter contained in the message text identified by %1, %2, %3.

MsgGet is used in conjunction with Error, Warning, MessageBox, and WinMessage to retrieve the corresponding message text from the Message Catalog. When a message set number less than 1 is supplied or the message is not in the catalog, the default message text is substituted. An example using MsgGet can be written as follows:

```
MsgGet(20003, 1, "Problem Id %1, is not resolved", MY_PROBLEM_ID)
```

MsgGet is not a function which can be run on its own. Without a preceding MessageBox, WinMessage, or Error, an 'Invalid Function Statement' message is issued by the syntax check. The MsgGet function call statement can be completed with a corresponding WinMessage:

```
WinMessage(MsgGet(20003, 1, "Problem Id %1, is not resolved",
MY_PROBLEM_ID));
```

14.4.2 MsgGetText

MsgGetText is similar to MsgGet. The key difference is that MsgGetText retrieves text from the Message Catalog without the message set and message number included in the message. MsgGetText cannot be implemented alone. It is used in conjunction with the other message functions such as WinMessage, MessageBox, Error, and Warning.

Refer to the code and accompanying message in figures 14.22 and 14.23. In the example, MsgGetText is used in conjunction with Warning. The message box does not display the message set and message number, which are 20002 and 1 respectively.

NOTE Fields and variables with a Number or Date/Time data type are converted to String when displayed in messages.

KEY POINTS

1. The `MessageBox` function is used to display messages from within PeopleCode programs.
2. The parameters used in `MessageBox` control the buttons, default button and icons in the message box window.
3. Messages can be defined by message set and message number, to the PeopleCode Message Catalog. These messages are then displayed by PeopleCode functions.
4. Messages can appear in various languages and are displayed in a chosen language, if the user profile is set up with the corresponding language code.
5. The `Error` and `Warning` functions work in conjunction with a `MsgGet` or `MsgGetText`, when displaying messages from the Message Catalog.
6. The difference between `Error` and `Warning` is that `Error` terminates the PeopleCode program and subsequent processing. `Warning` allows the user to continue after clicking the OK button.
7. `MsgGet` and `MsgGetText` retrieve messages from the Message Catalog. `MsgGetText` does not display the message set and message number.

CHAPTER 15

Embedded SQL

PeopleCode enables the developer to execute SQL statements for data access and update. SQL statements submitted from a `SQLExec` statement do not interact with the Application Processor. The statements are executed directly on the database server. Using `SQLExec` raises inherent issues such as security and the potential for runtime errors, which may not be identified by the PeopleCode editor.

15.1 WHEN TO USE EMBEDDED SQL

Application Designer and PeopleCode built-in functions offer several methods of retrieving and updating data. A panel can contain a field defined as a display control field. A display control field is used to establish a link to a related display field. The related display field usually contains some type of description loaded from a prompt or translation table. Retrieving values from a corresponding table requires that some type of common key is present. The subsequent data retrieval can only be completed when the panel is loaded into the buffer area.

Additional strategies that can be implemented to update and retrieve data include the `UpdateValue` function, which updates a value on a record, and `FetchValue`, which retrieves data from a record. `UpdateValue` and `FetchValue` are commonly used with scroll data. These functions, however, have their limitations.

15.2 THE SQLEXEC FUNCTION

When it is necessary to retrieve, update, or delete data outside of the common panel processor, PeopleCode provides the `SQLExec` function. `SQLExec` is a function which receives an SQL string, circumvents the Application Processor, and works directly with the database server to perform operations. Rather than selecting an entire row of data, as the Application Processor does, `SQLExec` only selects the field(s) specified. Not unlike other built-in functions, `SQLExec` has its limitations and drawbacks and it should be used with discretion. `SQLExec` can be a powerful ally to the developer when used correctly and efficiently.

15.2.1 SQLExec

`SQLExec` executes an SQL command passed as a string from a PeopleCode program. The SQL string can contain bind variables, subselects, and joins. Data elements appearing in a `Select` statement are returned to the PeopleCode program as output and can be stored in variables or record fields.

A `SQLExec` statement can be written as follows:

```
SQLExec("Select  NAME from PS_PERSONAL_DATA where EMPLID = :1",
&EMPLID, &NAME);
```

The preceeding example selects the name field from the HRMS PERSONAL_DATA record and stores it into a variable called `&NAME`. The `:1` represents a bind variable that contains an employee ID value used in the search criteria. Bind variables are the data elements referenced in the SQL string. Two types of bind variables exist: regular and inline. When regular bind variables are used, each requires a corresponding variable name which replaces the `:n` reference in the SQL string. These variables appear outside the double quotes. In the preceding example, the bind variable `:1` is substituted by the &EMPLID variable at runtime.

SQLExec is one function where unpredictable results can occur if rules are not followed. Because SQLExec bypasses the Application Processor and heads directly to the database, no evaluation of the SQL string contained in quotes is performed. Record fields used as inline bind variables or output variables are evaluated by the PeopleCode editor when they are not contained in the SQL string. When PeopleCode containing SQLExec statements are entered into the PeopleCode editor, any undefined record fields generate an error message during the syntax check or PeopleCode save operation. SQLExec statements containing inline bind variables are the exception. An incorrectly coded SQL statement that contains inline bind variables generates a runtime error message. Remember, the inline bind variables are enclosed within quotes. A previously undefined output variable such as &NAME is created at runtime and does not generate an error.

A SQLExec SELECT statement retrieves one row of data only. When multiple rows are selected, the data associated with the first row is the only data returned. What if the example above were rewritten as

```
SQLExec("Select  NAME from PS_PERSONAL_DATA where EMPLID <> :1",
&EMPLID, &NAME);
```

Only the first employee whose EMPLID does not match the contents of &EMPLID will have his/her name returned and stored in the &NAME output variable. The maximum number of output variables when using SELECT is 64.

With SQLExec, UPDATES, INSERTS, and DELETES can be performed, but can only be done in the following events:

- SavePreChg
- WorkFlow
- SavePostChg

SQLExec returns an optional Boolean. A value of True indicates that the function was successfully executed.

Let's now review a statement that contains a SQLExec function call. The PeopleCode is shown in figure 15.1. This statement is used in Problem Tracking to verify the MY_USER_ID field against the PeopleTool security record.

The SQLExec in this example is part of a nested If statement. In the example, we are verifying that the value contained in the record field MY_USER_ID exists on the PSOPRDEFN table. PSOPRDEFN is a Tools table which contains the operator definition (see chapter 3 for a description of the operator ID and class of operators). On the PSOPRDEFN table, we are comparing the OPRID column, which is the PeopleSoft operator ID, to the user ID entered into the Problem Tracking application panel. The bind variable represented as :1 is the first parameter to follow the SQL string. It is substituted with the contents of the field MY_USER_ID at runtime. If other bind variables were used, they would follow MY_USER_ID in the order in which they appeared in

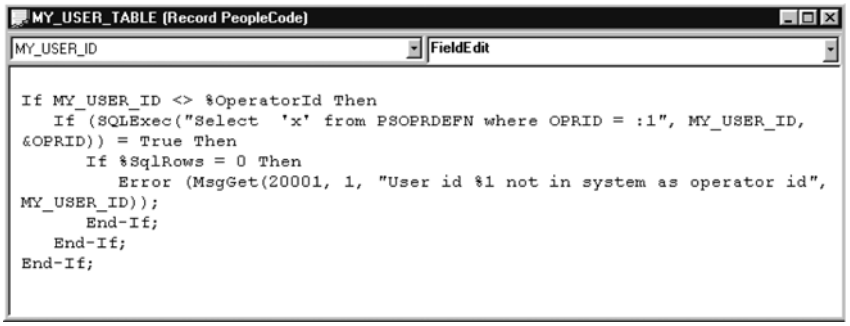


Figure 15.1 SQLExec statement

the SQL string and would subsequently be identified as `:2`, `:3`, and so on as required. A bind variable cannot reference a `LONG` data type.

In the example, the next parameter that follows MY_USER_ID is an output variable. This can also be a record field. The example uses an output variable named `&OPRID` which does not require pre-definition. Upon closer examination, however, the output variable `&OPRID` does not contain the results of the `Select`. In this example we use a convention that substitutes a literal for the OPRID field. This is done to verify that the value contained in MY_USER_ID exists on the table. Selecting the actual value into `&OPRID` is redundant but acceptable. Any references to `&OPRID`, however, are not accurate if we select the literal 'x' into the variable.

`SQLExec` returns an optional Boolean. In the example, the code checks for a value of `True` to indicate a successful function call. In the context of this example we are using the Boolean return value to determine if the next statement will be executed. In the example, the function return value is verified before executing the next statement. An alternative method of writing the `SQLExec` portion is:

```
If MY_USER_ID <> %OperatorId Then
   If (SQLExec("Select  'x' from PSOPRDEFN where OPRID = :1", MY_USER_ID,
&OPRID)) Then
      If %SqlRows = 0 Then
         Error (MsgGet(20001, 1, "User id %1 not in system as operator id",
MY_USER_ID));
      End-If;
   End-If;
End-If;
```

Because `SQLExec` returns a Boolean value, the comparison operator is not necessary. People who are new to development may find this convention somewhat strange. The `If` statement evaluates the expression in parenthesis. When the expression is `True`, any subsequent statements are executed.

The next statement is interesting: `SQLExec` uses the system variable `%SQLRows` to identify the number of rows impacted by the most current `SQLExec`. The number

of rows affected by an UPDATE, INSERT, or DELETE statement is reflected in the %SQLRows system variable. A SELECT statement, however, returns zero when no rows are selected and returns a non-zero value if one row is selected.

NOTE When performing a Select, the non-zero value returned in %SQLRows does not reflect the total number of rows selected. The non-zero value is a return code, not the actual number of rows selected.

15.3 USING INLINE BIND VARIABLES

Refer to the example in figure 15.1. When an inline bind variable is used to represent the field MY_USER_ID, the PeopleCode appears as shown in figure 15.2.

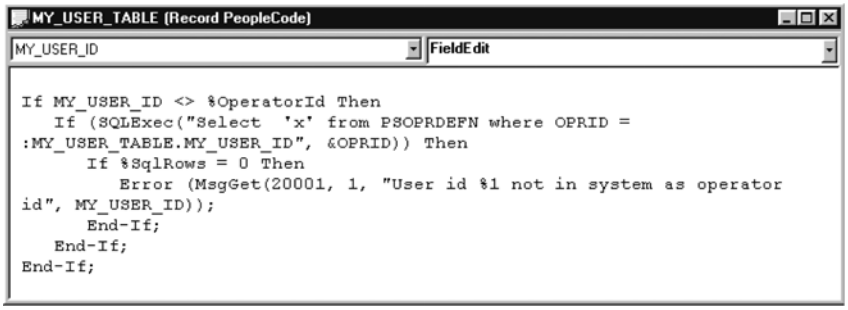


Figure 15.2 Inline Bind variables

Figure 15.3 Runtime error using incorrect bind variable

The inline bind variable requires full record field qualification. A runtime error message is issued when the bind variable is coded as :MY_USER_ID. An error message however is not issued during the PeopleCode editor session. No error is generated because the inline bind variable appears within the SQL string, and no edits are performed against it until runtime. When regular bind variables are used, the PeopleCode editor verifies that the correct record and field combination are defined to the Application Designer. An example of a runtime error message is displayed in figure 15.3.

Conversely, if we erroneously code the inline bind variable illustrated in figure 15.2 as :OUR_USER_TABLE.MY_USER_ID, the PeopleCode editor does not generate an error during a syntax check or when the code is saved. Instead, we receive a runtime error as illustrated by figure 15.4.

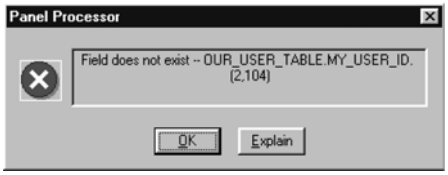


Figure 15.4 Incorrect record field used as inline bind variable

We should consider several items when using inline bind variables. The PeopleCode editor does not check for incorrect record names or field names referenced as inline variables in the SQL string. These names are resolved at runtime. If you are part of a development team using custom tables, the possibility exists that someone may rename a table or a field referenced as an inline bind variable. In the example in figure 15.2, if the table were actually renamed to OUR_USER_TABLE by someone who thought it more appropriate than MY_USER_TABLE, the PeopleCode would produce a runtime error. This assumes that, after the renaming was accomplished, syntax checking was performed against the PeopleCode. Similarly, a PeopleTools upgrade may also contain table name changes bundled into the upgrade. If you have developed custom code, the references to these tables or fields within an SQL string are not updated automatically by the upgrade process. Any syntax checking will flag record and fields that do not exist, provided they are not contained in the SQL string as inline bind variables.

15.4 DATES AND META-SQL

There are platform limitations to using embedded SQL particularly for Date/Time values. Each platform has its own unique method when it comes to handling dates. The methods are not dramatically different, but the differences are enough so that in order to remain platform independent, PeopleTools uses its own date formatting. In PeopleCode programs, the Application Processor converts data for use in PeopleTools applications. Data that may appear different from one platform to another can include dates and data types such as `LONG`. The Application Processor performs these conversions when data are loaded into buffers for input processing or when the data are moved from the buffer to the database. The `SQLExec` statement does not perform this type of data conversion. In essence, when using `SQLExec`, what is read in as a result of a `Select` is what appears in the output value.

The Application Processor stores dates as `YYYY-MM-DD`. When a `SQLExec` loads a date value stored on the database as `DD-Mon-YYYY`, a subsequent comparison between the two dates will always result in False.

One method that enables us to get around these platform specific issues is the `Meta-SQL` function. `Meta-SQL` functions are imbedded into statements that receive an SQL string such as `SQLExec`. Most `Meta-SQL` functions can be categorized into two types, either as an in function or an out function.

There are however some functions such as `TrimSubStr` and `SubString` that serve dual roles.

At runtime, `In` functions containing `UPDATE`, `SELECT`, or `INSERT` statements extend to become platform specific SQL. These functions generate SQL statements

containing variables that are passed to the database. In functions can be used, for example, when a date is used in the WHERE clause of a Select or Update statement. In functions can also be used when a date is sent to the database via an Insert statement. Out functions also extend at runtime to become platform-specific SQL which appear in Select statements.

Definitions of Meta-SQL functions are listed in appendix E.

The examples below compare a SQLExec statement using the platform-specific ORACLE/DB2 substring against the PeopleCode platform independent %Substring Meta-SQL. In both examples we are selecting the first ten characters from the NAME field of the PERSONAL_DATA record. The first example uses the platform-specific SUBSTR function:

```
SQLExec("Select  SUBSTR (NAME,1,10) from PS_PERSONAL_DATA
  where EMPLID = :1", &EMPLID, &NAME)
```

While this example uses the %Substring Meta-SQL function:

```
SQLExec("Select  %Substring(NAME,1,10) from PS_PERSONAL_DATA
  where EMPLID = :1", &EMPLID, &NAME)
```

The statements are similar but the benefit of the statement in the second example is platform independence. The SUBSTR function uses the same basic parameters as the Meta-SQL. The argument string, starting position, and length are even in the same order. Nothing is as easy as it appears, however, and this is no exception. Suppose we are using ORACLE specific functions and the MIS management has decided to migrate functionality to SQLBase. If we were to use platform-dependent functions, our code would appear as:

```
SQLExec("Select  @SUBSTRING(NAME,0,10) from PS_PERSONAL_DATA
  where EMPLID = :1", &EMPLID, &NAME)
```

We can immediately see that the function is @SUBSTRING. Another and more potentially risky result is that, when using the @SUBSTRING function, the first character in the string begins at position 0. The ORACLE, DB2, and Meta-SQL start at position 1. Without careful analysis, these types of changes can lead to major problems later on.

Let's look at another example using dates. With no edits, dates are selected in formats, depending on the database type. Some databases select dates as YYYYMMDD, and others select dates as DD-MON-YYYY. In the following example, we are looking for a total of all employees in PERSONAL_DATA who have a BIRTHDATE <= January 1st 1970. The count is placed into a variable called &SUM. A database specific piece of code may appear as:

```
&COMPARE_DATE = "1970-01-01";
SQLExec("Select  COUNT(*) from PS_PERSONAL_DATA  where BIRTHDATE <=  TO_
DATE(:1,'YYYY-MM-DD')", &COMPARE_DATE, &SUM);
```

Using an `In` Meta-string, we can obtain platform independence. The example below uses the `%DateIn` function and receives a date in the format `YYYY-MM-DD`.

```
&COMPARE_DATE = "1970-01-01";
SQLExec("Select  COUNT(*) from PS_PERSONAL_DATA  where BIRTHDATE <=
%DateIn(:1)", &COMPARE_DATE, &SUM);
```

15.5 SECURITY AND MAINTENANCE CONSIDERATIONS

When using `SQLExec` with inline bind variables, we need to keep in mind that future upgrades may rename tables or fields. Syntax checks of inline bind variables will not indicate this type of discrepancy. Inconsistencies can also occur when a custom table is modified and any corresponding code does not reflect the change.

More importantly, we need to know that the use of `SQLExec` has the potential to allow the end-user to update or delete data. Assuming the user has database privileges, coding `SQLExec` without giving consideration to security can result in a user unknowingly changing data. This can most likely occur when the `SQLExec` statement has been coded incorrectly, without verifying security. Perhaps the `SQLExec` updates data to which the user would not normally have access.

KEY POINTS

1. `SQLExec` is a function that allows the developer to execute SQL statements directly on the database server.
2. Bind variables can be passed two ways: regular and inline.
3. `Meta-SQL` enables `SQLExec` to work with `Date`, `Time`, and `String` parameters. This allows some level of platform independence.
4. Of utmost importance is that the use of `SQLExec` bypasses the standard PeopleSoft security. The security in place may limit the user access to specific data. Unless `SQLExec` code mimics the security functionality, improper use of `SQLExec` can result in a security breach.

CHAPTER 16

Working with scrolls

This chapter discusses the methods used to process panels that contain scrolls. Data consisting of multiple rows may occupy the same panel and include various primary record definitions. Knowledge of how system buffers are processed is necessary for panels or panel groups containing multiple occurs levels, because the Application Processor utilizes PeopleCode in the same order. Not every panel contains or requires a scroll bar. For panels that do contain scroll bars, specific terminology, functions, and methods are often used to reference data in the panel buffers. Topics aimed at providing an understanding of how to use PeopleCode in multiple occurs levels include:

- the relationship between records on different scroll levels
- the order in which the system processes buffers
- PeopleCode functions used with scrolls
- the SQL string within scroll functions

In part 2, Advanced Panel Design Features, we discussed how to add scroll bar objects to panels. A panel containing scroll bars may include multiple occurs levels and occurs count. At the PeopleCode level, a large amount of code can be devoted to working within multiple occurs levels. Multiple occurs levels impact PeopleCode execution because the order in which the Application Processor handles scroll buffer areas and the manner in which PeopleCode is written must work together. Understanding the relationship between records at varying occurs levels is important for writing efficient code.

16.1 PARENT/CHILD RELATIONSHIP

According to PeopleTools terminology, a record at the highest level in a panel is referred to as the level zero record. From a multiple level perspective, a panel containing only one record and no scroll bars is elementary. On a panel containing one scroll bar, two occurs levels exist: zero and one. What we have at occurs level zero is one or more record definitions that contain one or multiple fields, used to link data in subsequent levels. The level zero record is considered the Parent row, and level one is the Child. When data buffers are processed, level zero records are handled before level one records. When a panel contains two scroll bars, the level zero records are processed first and are then followed by one row of level one data. When there are two occurrence levels, the level one record is parent to level two. After a single level one record is processed, the system then processes level two records. The cycle is then repeated; the next level one row of data are processed, then followed by all associated level two records.

Table 16.1 illustrates parent and child keys:

Table 16.1 Relationship between Parent and Child rows

SETID	OPRCLASS	Location	Emplid
USA	ALLPANLS	001	6601
USA	ALLPANLS	001	7705
USA	ALLPANLS	001	6603
USA	ALLPANLS	002	8101
USA	ALLPANLS	002	8102
USA	ALLPANLS	002	8105
USA	ALLPANLS	003	8201
USA	ALLPANLS	003	8651
USA	ALLPANLS	003	8773

The parent and child keys are `SETID`, `OPRCLASS`, and `Location`. An additional child key is `Emplid`.

A selected parent row contains three key values. Child rows also contain the same three key values plus an additional key. In this example `EMPLID` is the additional child

key. A parent key containing `'001'` as the third key value may have one or more child rows that contain the same key values. On a panel containing two scrolls representing the data illustrated in table 16.1, when parent key `001` is encountered, the child keys `6601`, `7705`, and `6603` are processed before parent key `002` and its corresponding child key values `8101`, `8102`, and `8105`.

When scroll bars are defined with `AutoSelect`, all data are retrieved with one `Select` statement, regardless of the number of scrolls. The `Select` is performed after a search key is chosen. The Application Processor then manages the buffers by processing level zero records first, then all subsequent levels as discussed previously.

TIP Level zero fields are usually based on search records.

Using the knowledge obtained in part 2, it is possible to build a panel containing two scroll bars and no PeopleCode. We could define a level zero record, a level one record, and a level two record. Without PeopleCode, however, the functionality is limited to certain panel processor functions and to end-user actions such as `Insert Row` (F7) and `Delete Row` (F8). At this basic level, we could define a panel, add scroll bars, and establish the parent/child relationships. When the panel is processed, any scroll areas would be populated automatically. There are circumstances, however, under which we, as the developers, would have to programmatically control the processing of panel scroll areas. Our work becomes complex and, therefore, more interesting when PeopleCode is added to enhance the processing behind panel scroll areas.

The objective of this chapter is to enable the reader to develop panels using scroll bars and, more importantly, to add the necessary PeopleCode that will interact with data contained in these scrolls. This includes the following items:

- loading data into a scroll
- removing data from a scroll
- removing scroll data from a database table
- determining the number of records in a scroll
- retrieving or updating data in a scroll buffer area

To begin our tour of scroll functionality, we will use two panels containing scrolls. The concept behind the panels is to link operator security classes and office locations that are on the LOCATION_TBL record. We then link the employees in the specified locations to the operator class/location combination. Figure 16.1 illustrates the Operator Class/Location panel, and figure 16.2 is the panel that links the operator class/locations to employee data.

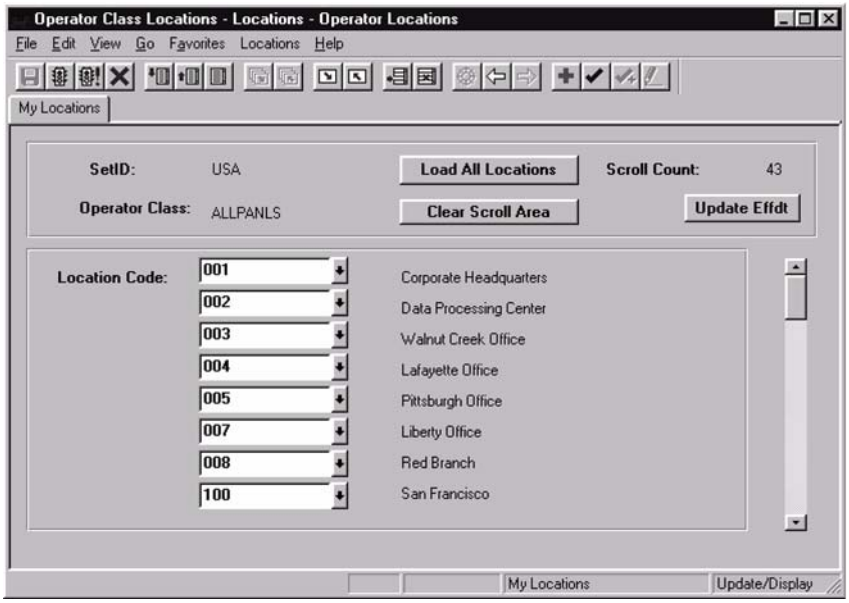


Figure 16.1 Operator Class/Location panel

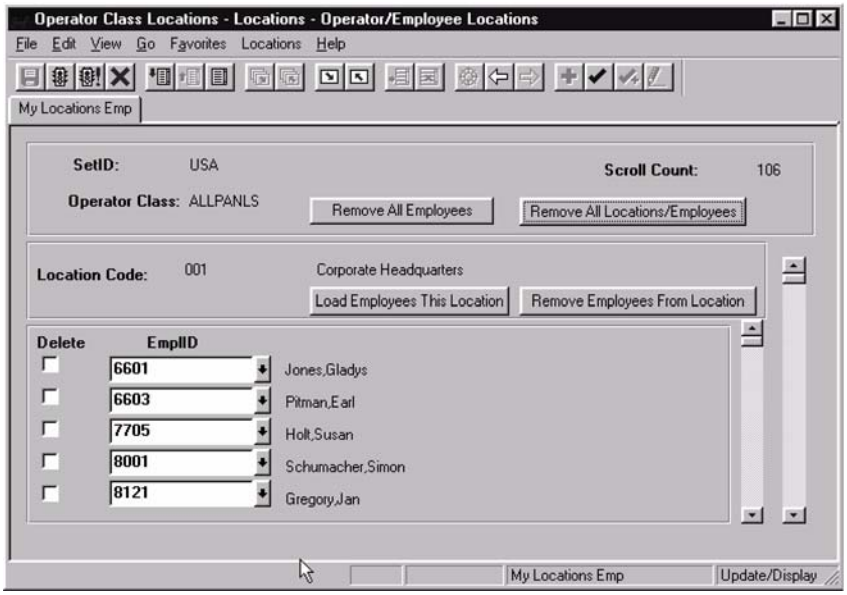


Figure 16.2 Link employees to operator/class locations

As we can see by the illustrations, one panel contains a single scroll bar, and the other panel contains two. When using functions that operate on scroll areas, the number of parameters passed to these functions varies based on the number of scrolls. The maximum number of scroll levels permitted is three. When adding a scroll to a panel, a scroll at level 3 cannot be defined without a scroll at level 2. Similarly, a scroll at level 2 cannot be defined without a scroll at level 1. A hierarchy among keys should also be followed. The level 2 record requires the same key fields as level 1 in addition to its own unique key. The level 1 record requires the level 0 key fields, with the addition of a unique key value.

Referring to figure 16.1, we can see several record definitions in the panel because it is a panel with one scroll containing two primary records. The level 0 primary record is MY_LOCATION_HDR. This record contains two keys, SETID and OPRCLASS. The primary record at level 1 is MY_LOCATIONS. This record is a child of MY_LOCATION_HDR. The order and level of fields for the operator class/locations panel is illustrated by figure 16.3. MY_LOCATION_HDR is considered the primary scroll record, because the keys from this table control the data selected into subsequent scrolls.

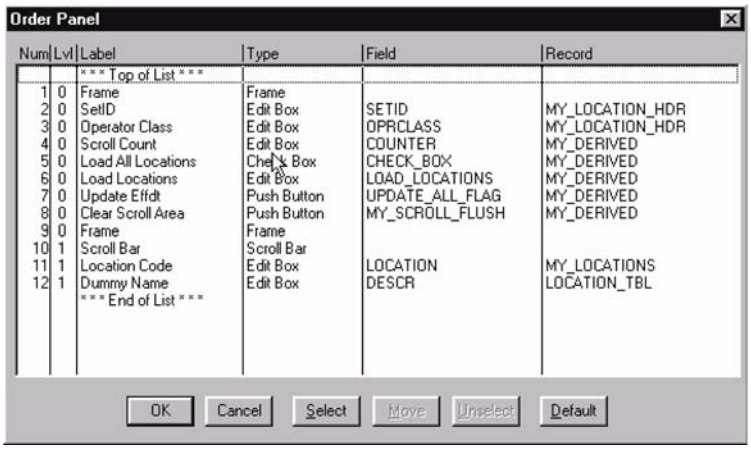


Figure 16.3 Order and level of fields for Operator Class/Location panel

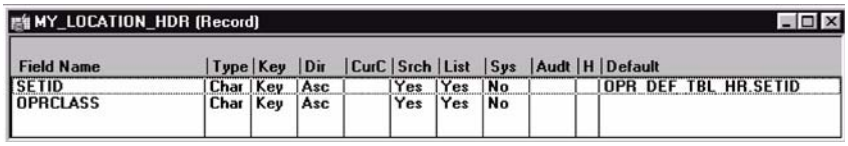
---

NOTE For each scroll level, only one primary scroll record can exist. Other records can be those of related display fields and Derived/Work fields.

The illustration in figure 16.3 also identifies an additional record at level 0, MY_DERIVED. This record contains the PeopleCode linked to the push buttons on the panel. At level 1, we have the record MY_LOCATIONS and the location table

(LOCATION_TBL). The location table is used to retrieve the location description and effective date.

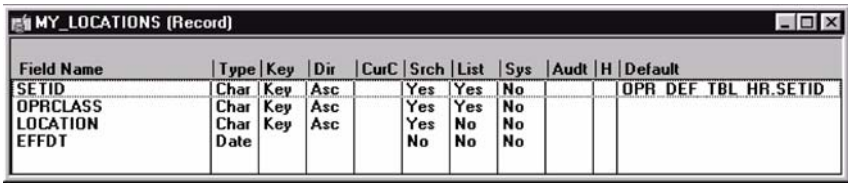
The record definitions and key fields for the three primary records used in figures 16.1 and 16.2 are shown in figures 16.4, 16.5 and 16.6.



MY_LOCATION_HDR (Record)

Field Name	Type	Key	Dir	CurC	Srch	List	Sys	Audt	H	Default
SETID	Char	Key	Asc		Yes	Yes	No			OPR_DEF_TBL_HR.SETID
OPRCLASS	Char	Key	Asc		Yes	Yes	No			

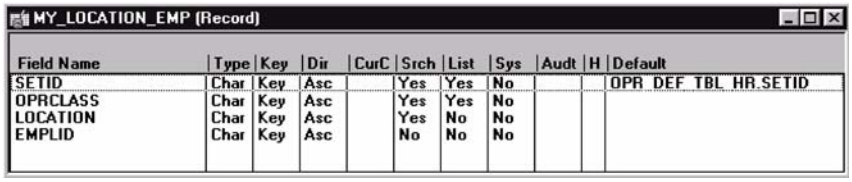
Figure 16.4 Fields for MY_LOCATION_HDR



MY_LOCATIONS (Record)

Field Name	Type	Key	Dir	CurC	Srch	List	Sys	Audt	H	Default
SETID	Char	Key	Asc		Yes	Yes	No			OPR_DEF_TBL_HR.SETID
OPRCLASS	Char	Key	Asc		Yes	Yes	No			
LOCATION	Char	Key	Asc		Yes	No	No			
EFFDT	Date				No	No	No			

Figure 16.5 Fields for MY_LOCATIONS



MY_LOCATION_EMP (Record)

Field Name	Type	Key	Dir	CurC	Srch	List	Sys	Audt	H	Default
SETID	Char	Key	Asc		Yes	Yes	No			OPR_DEF_TBL_HR.SETID
OPRCLASS	Char	Key	Asc		Yes	Yes	No			
LOCATION	Char	Key	Asc		Yes	No	No			
EMPLID	Char	Key	Asc		No	No	No			

Figure 16.6 Fields for MY_LOCATION_EMP

NOTE Only one row of data for each level 0 record is allowed on a panel. Other rows are displayed on a list box.

Now that we are better acquainted with the panels, records, and parent/child relationships involved in the scroll demonstration, we can begin to review the functions and apply them to the panels in figures 16.1 and 16.2.

Navigation: Locations →Operator Locations →Add

Figure 16.7 Establish the operator class header

From a functional perspective, when an operator class is linked to one or more locations, a location header record has to be initially established. We can use the menu to select the `SETID` and `OPRCLASS`.

After the operator class header is established, the PeopleCode behind the push button is ready to load data into the scroll area (figure 16.8).

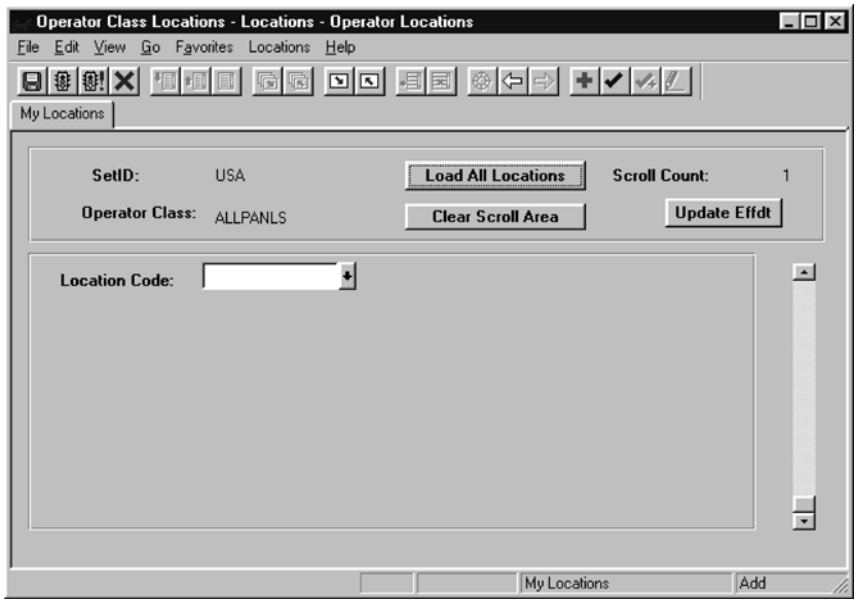


Figure 16.8 Initial Operator Class/Location panel

The panel shown in figure 16.8 refers to information for MY_LOCATION_HDR but contains no data in the scroll area referencing MY_LOCATIONS. The objective here is to load all location codes from the location table for this particular `SETID`. Several methods can be used to accomplish this task. One method involves the use of PeopleCode Scroll functions.

16.2 PEOPLECODE FUNCTIONS USED WITH SCROLLS

The first function that will be applied is `ScrollSelect`. Functions that operate on data contained in a scroll use the `ScrollPath` to reference the individual row or scroll levels at which the functions are targeted. `ScrollPath` defines the records at

each scroll level as well as the target record name. When we reference a scroll at level 1, ScrollPath is comprised of the target record only. A reference to data at scroll level 2 requires specification of the level 1 record and the target record name. When referring to rows or data on scroll level 3, level 1 and level 2 records are required, in addition to the target record name.

16.2.1 ScrollSelect

The ScrollSelect function selects records from a table and loads them into the scroll buffer area of a panel. In terms of Parent/Child relationships, ScrollSelect chooses all corresponding child rows and inserts them under the next higher level row. The function requires the specification of the target scroll area, a source record from which to select rows and an optional SQL string. The parameters passed to ScrollSelect vary based on the scroll level at which the function is targeted:

Level 1

```
ScrollSelect (1, RECORD.target_recname, RECORD.sel_recname);
```

Level 2

```
ScrollSelect (2, RECORD.level1_recname, RECORD.target_recname,
RECORD.sel_recname);
```

Level 3

```
ScrollSelect (3, RECORD.level1_recname, RECORD.level2_recname,
RECORD.target_recname, RECORD.sel_recname);
```

In addition to the parameters required to reference data at the various scroll levels, the optional SQL string and Turbo parameters can be specified. (Refer to appendix E for syntax description of ScrollSelect.) Let's apply ScrollSelect to the panels presented in figures 16.1 and 16.2.

Example 1

Figure 16.8 contains a push button—Load All Locations. We can use ScrollSelect to load all location codes from the location table for the SETID contained in the record MY_LOCATION_HDR. The code using ScrollSelect is illustrated in figure 16.9.

The code in figure 16.9 first verifies that the panel name is MY_LOCATIONS. This is necessary because the code is executed from a Derived/Work record (MY_DERIVED), which can be used on different panels. The DeleteRow statement is used to delete rows from the scroll and database table. In this example, (DeleteRow is used in a loop. Rows are processed from high to low, because rows are renumbered each time they are deleted.) The ScrollSelect parameters identify that the target

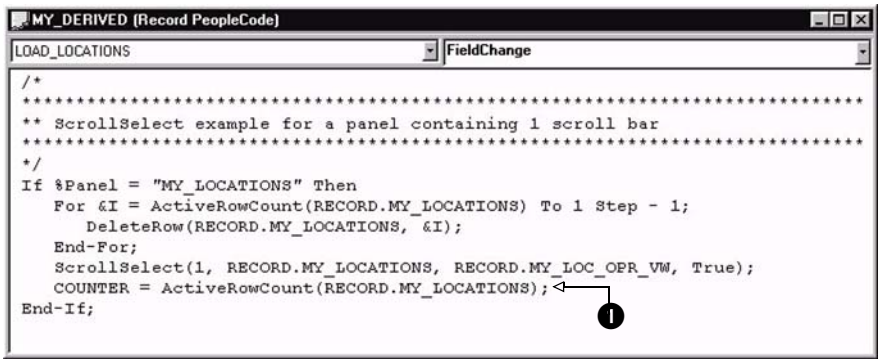


Figure 16.9 ScrollSelect at level 1 scroll

scroll area is at level 1. Based on the scroll level, `target_recname` and `sel_recordname` parameters are required. `MY_LOCATIONS` is the target record name and the select scroll area. Data are retrieved from `MY_LOC_OPR_VW`, which is a view that selects locations with the most current effective date from the LOCATION_TBL record. This parameter can be the same as the target record name, but in this example, we are selecting from a view and loading the selected fields into a different target record. The COUNTER field, ❶, contains the number of active rows in the scroll area and is reflected on the panel. The `True` parameter at the end of the function call indicates that we are using the `Turbo` feature. When specified in Scroll functions such as `ScrollSelect` and `ScrollSelectNew`, `Turbo` improves performance of `ScrollSelect`.

(Refer to figure 16.1 for an illustration of how the panel appears after the `ScrollSelect` PeopleCode is executed.)

Example 2

The next panel links employees to the operator class/locations.

The panel illustrated by figure 16.10 contains a push button—Load Employees This Location. The code behind this button is used to populate the scroll area with employee IDs that have a current location code equal to the value of the location on the current level 1 scroll. The code to accomplish this task on scroll level 2 is shown in figure 16.11.

In addition to `ScrollSelect`, other functions and statements are used to accomplish the task of loading employee data for the location at scroll level 1. The code verifies the panel name is MY_LOCATIONS_EMP. This is necessary because the code is executed from a Derived/Work record and can be used on different panels. Conceptually, the code used to load the scroll on MY_LOCATIONS and MY_LOCATIONS_EMP panels can be localized on the same record and fieldname in the Derived/Work record. The `DeleteRow` statement is used to delete rows from the

Navigation: Locations →Operator/Employee Locations

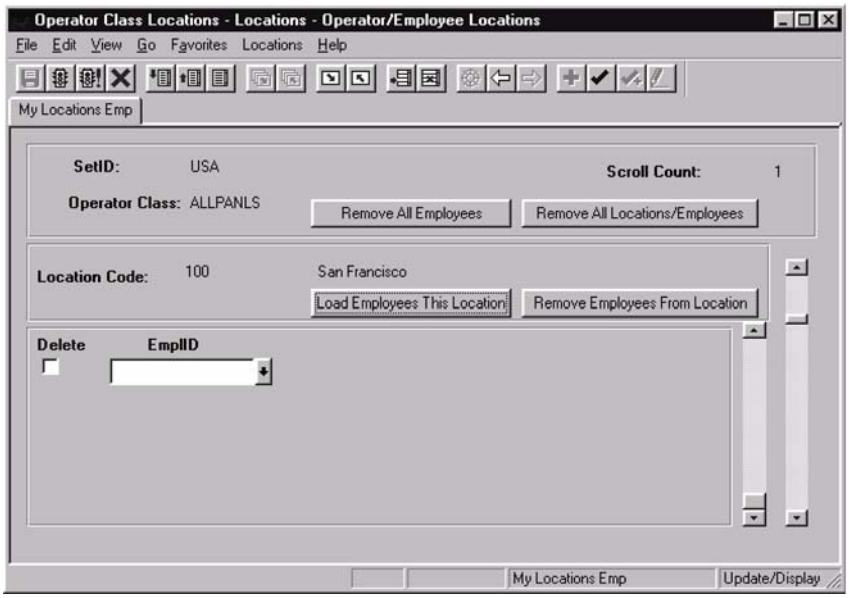


Figure 16.10 Panel to link operator class/locations and employees

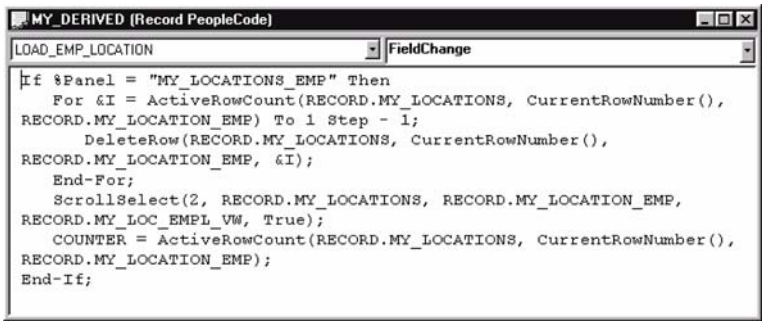


Figure 16.11 ScrollSelect at level 2 scroll

scroll area and database table. The `DeleteRow` is used in a loop. Rows are processed from high to low, because rows are renumbered each time they are deleted. `DeleteRow` will be discussed later in this chapter. The `ScrollSelect` parameters identify that the target scroll area is at level 2. This scroll level will contain the selected employee IDs. Because the target record is at level 2, the level1_recname parameter is required. In the example the level 1 record name is coded as RECORD.MY_LOCATIONS. The target record is MY_LOCATION_EMP and resides

at level 2. In the example the select record (sel_recordname) is represented by a view. A view is used to extract the most current effective-dated Job rows and join them with the corresponding location table entry. The optional Turbo parameter is set to TRUE so that performance of ScrollSelect can be improved. Because the potential to load many rows of employee IDs at scroll level 2 exists, the use of this parameter is vital. The COUNTER field contains the number of active rows in the scroll area. This count is reflected on the panel as is the value passed by the ActiveRowCount function. This function will be discussed later, but it is worthwhile to mention that its parameters are based on the target scroll level referenced.

A function similar to ScrollSelect is ScrollSelectNew.

16.2.2 ScrollSelectNew

ScrollSelectNew resembles ScrollSelect except that ScrollSelectNew marks records as New when they are loaded into the scroll area. During save processing, these records are automatically added to the database. ScrollSelect is used to select pre-existing rows into a scroll area. Because ScrollSelect does not mark rows as New, some other type of activity is required to enable the save button. A DeleteRow used in combination with ScrollSelect enables the save button for new rows. ScrollSelectNew requires the specification of the target scroll area, a source record from which to select rows, and an optional SQL string. The parameters passed to ScrollSelectNew vary based on the scroll level at which the function is targeted.

Level 1

```
ScrollSelectNew (1, RECORD.target_recname, RECORD.sel_recname);
```

Level 2

```
ScrollSelectNew (2, RECORD.level1_recname, RECORD.target_recname,
RECORD.sel_recname);
```

Level 3

```
ScrollSelectNew (3, RECORD.level1_recname, RECORD.level2_recname,
RECORD.target_recname, RECORD.sel_recname);
```

In addition to the parameters required to reference data at the various scroll levels, the optional SQL string and Turbo parameters can be specified.

Example

The code in figure 16.12 applies ScrollSelectNew to the level 1 scroll that appears on the panel MY_LOCATIONS.

As we can see by the sample code, ScrollSelectNew is essentially the same as ScrollSelect. The parameters and their use are identical. In the example,

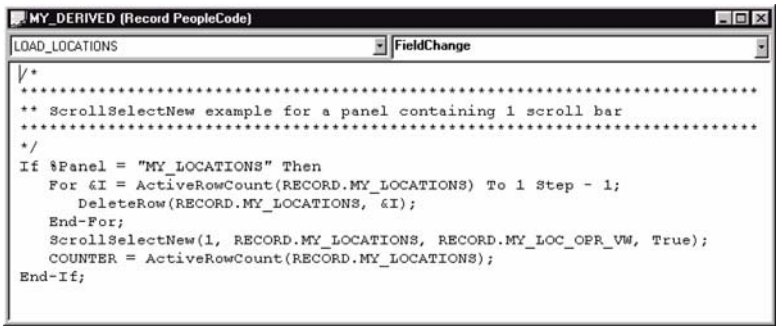


Figure 16.12 Using ScrollSelectNew at level 1

`ScrollSelectNew` operates on target scroll level 1. The key difference is that the Location Table entries matching the `SETID` at level 0 are loaded into the scroll buffer and are marked as `NEW`. During save processing, they are added to the database.

The panel in figure 16.1 can be used to illustrate the results of `ScrollSelect-New` if it were used in place of `ScrollSelect`.

This panel also contains additional characteristics due to its ability to utilize the F7 and F8 function keys.

Locations can also be added by using the F7 Insert Row key and the Location prompt table, rather than using the "Load All Locations" push button. The F8 Delete Row key can also be used to remove unwanted rows. To give a real world example, let's say that the user has elected to load all locations automatically, using the push buttons, but then decides to enter the locations manually. At this point, the user can cancel out and start again. We can however, make the process more efficient by providing the ability to clear the scroll area before it is saved. This can be done using the `ScrollFlush` function.

16.2.3 ScrollFlush

`ScrollFlush` is used to remove records from a target scroll area. The function requires the specification of the target scroll area as the `ScrollPath`. The parameters passed to `ScrollFlush` are based on the scroll level from where the rows are to be removed.

Level 1

```
ScrollFlush (RECORD.target_recname);
```

Level 2

```
ScrollFlush (RECORD.level1_recname, level1_row, RECORD.target_recname);
```

Level 3

```
ScrollFlush (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname);
```

Rows flushed from the target scroll area are not removed from the database.

Example 1

Figure 16.8 contains a push button labeled "Clear Scroll Area." The PeopleCode is shown in figure 16.13. When activated, the PeopleCode behind the push button clears the target scroll area using `ScrollFlush`.

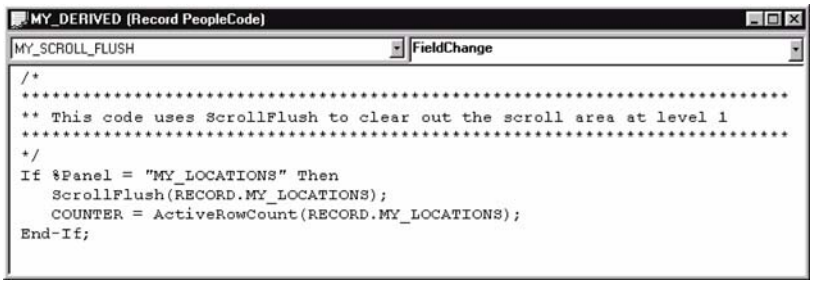


Figure 16.13 ScrollFlush at level 1

A field named MY_SCROLL_FLUSH is added to MY_DERIVED and then placed on the panel. Because MY_DERIVED is a work record, there is no need to alter or recreate the table after the field is added to the record. The code to accomplish the `ScrollFlush` is placed in the `FieldChange` event. The example takes one parameter because we are clearing rows at scroll level 1. As a result, only the target record name is specified.

Example 2

Figure 16.10 contains a push button labeled "Remove Employees From Location." The button can be used to clear out the scroll area after employee data has been loaded. The PeopleCode utilizes `ScrollFlush` at the level 2 scroll. The example is illustrated in figure16.14.

A review of the PeopleCode in figure 16.14 indicates the panel name is verified as MY_LOCATIONS_EMP. The `ScrollFlush` parameters use level1_recname. This parameter is required because the target record is at level 2. The example specifies the level 1 record name as `RECORD.MY_LOCATIONS`. `ScrollFlush` is targeted at scroll level 2 and as a result, the level1_row parameter is required. The `CurrentRowNumber` function is used to identify the row number at scroll level 1. The target record is MY_LOCATION_EMP, which resides at level 2.

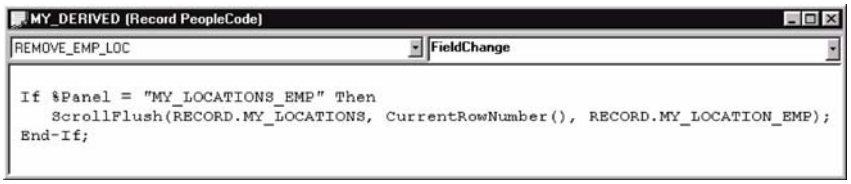


Figure 16.14 ScrollFlush at Scroll level 2

NOTE The `ScrollFlush` function does not delete records from the database. It removes them from the target scroll and related buffer areas only.

16.3 ADDITIONAL SCROLL FUNCTIONS

At this point, the reader should have a better understanding of scrolls and some of the functions used to process them, as discussed in the previous sections. The following additional scroll functions use parameters similar to those discussed previously.

It is sometimes necessary to identify the number of rows in a scroll area. PeopleCode uses several functions to count records in a target scroll area. Several count functions are available because data marked for deletion may not be required in some routines. `ActiveRowCount` is one such function:

16.3.1 ActiveRowCount

`ActiveRowCount` returns a number representing the sum of active rows in a given scroll area. Records marked as deleted are not included in the count. `ActiveRowCount` is often used when we are looping and examining each row in the target scroll area. The parameters used by `ActiveRowCount` are based on the scroll level in which a count of active rows is required.

Level 1

```
ActiveRowCount (RECORD.target_recname);
```

Level 2

```
ActiveRowCount (RECORD.level1_recname, level1_row, RECORD.target_recname);
```

Level 3

```
ActiveRowCount (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname);
```

Example 1

The Operator Class/Location panel in figure 16.1 can be used to illustrate `ActiveRowCount` at scroll level 1. The panel contains a label named "Scroll Count,"

used to indicate the number of non-deleted rows in the scroll area. The COUNTER field on MY_DERIVED contains the return value from the ActiveRowCount function that appears on the panel.

The use of ActiveRowCount to display the number of rows in a scroll requires that it be placed strategically in various events, in order for the count to reflect inserts and deletes correctly.

The two panels used in the Operator Class/Location application can use ActiveRowCount in the following events or actions:

- RowInit
- ScrollSelect/ScrollSelectNew
- ScrollFlush
- RowInsert (F7)
- RowDelete (F8)

The example below can be applied to the panel in figure 16.1, which contains one scroll level. Each time any push button is activated, the "Scroll Count" field is updated. The field named COUNTER reflects the number of active rows in the scroll area.

```
COUNTER = ActiveRowCount(RECORD.MY_LOCATIONS);
```

The preceding code can be applied to scroll level 1. At scroll level 2, the following code returns the number of active rows. The count is applied to the panel illustrated in figure 16.2.

```
COUNTER = ActiveRowCount(RECORD.MY_LOCATIONS, CurrentRowNumber(),
RECORD.MY_LOCATION_EMP);
```

Example 2

The use of ActiveRowCount can be applied during loop processing. The example below uses ActiveRowCount when using the DeleteRow function. In the example, the variable &I initially contains the number of active rows, which enables the loop to work from the highest to the lowest row in the scroll.

```
For &I = ActiveRowCount(RECORD.MY_LOCATIONS) To 1 Step - 1;
   DeleteRow(RECORD.MY_LOCATIONS, &I);
End-For;
```

Functions such as ActiveRowCount and ScrollFlush require a level row number as part of the ScrollPath. A scroll area can contain one or many rows. Each row has a number associated with it which indicates its place in the scroll area. The number can be used to identify a parent row to its corresponding child rows. A function that returns the number associated with a row in a scroll area is CurrentRowNumber.

NOTE When no data appears in a scroll, ActiveRowCount returns 1. When there is one row, the function still returns 1.

16.3.2 CurrentRowNumber

The CurrentRowNumber function is used when it is necessary to identify the row number of the current row in a scroll area. The function takes a parameter which represents the level where the row number is retrieved. When the level parameter is not specified, the function uses the current scroll level from where the function is called as the default level. CurrentRowNumber is sometimes used with ActiveRowCount to limit the number of times a loop is processed based on the active rows in the scroll area.

Example

The following example uses CurrentRowNumber during execution of the ActiveRowCount and DeleteRow functions. When used in conjunction with ActiveRowCount and DeleteRow, CurrentRowNumber returns the parent row number as a path to the child row MY_LOCATION_EMP. In this context, the parent row may reference one or more child rows.

```
For &I = ActiveRowCount(RECORD.MY_LOCATIONS, CurrentRowNumber(),
   RECORD.MY_LOCATION_EMP) To 1 Step - 1;
    DeleteRow(RECORD.MY_LOCATIONS, CurrentRowNumber(),
      RECORD.MY_LOCATION_EMP, &I);
End-For;
```

16.3.3 DeleteRow

A number of PeopleCode programs imitate user actions performed using toolbar icons or function keys. The DeleteRow function can be used to delete records from a scroll area and database.

The DeleteRow function enables rows to be deleted by a PeopleCode program. The function triggers the RowDelete event that mimics the F8/Delete Row operation. DeleteRow removes records from the target scroll as well as from the database. DeleteRow requires the specification of the ScrollPath and the target row number to delete. The parameters passed to DeleteRow are based on the scroll level and target record number from where rows are to be removed. Using the DeleteRow function at various levels can be written as follows:

Level 1

```
DeleteRow (RECORD.target_recname, target_row);
```

Level 2

```
DeleteRow (RECORD.level1_recname, level1_row, RECORD.target_recname,
target_row);
```

Level 3

```
DeleteRow (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname, target_row);
```

NOTE `DeleteRow` cannot be called from the same scroll level as that of the target scroll area.

Example 1

The panel in figure 16.10 used to link operator classes to employees contains a button labeled "Remove All Employees." When activated, the PeopleCode removes employee data only and removes it from every record at scroll level 1. The code utilizes two loops. The outer loop retrieves the number of active rows for scroll level 1. The inner loop references the child rows that contain employee data and deletes them. The code to accomplish this task is shown in figure 16.15. Data for the parent record MY_LOCATIONS remains intact after the code completes execution.

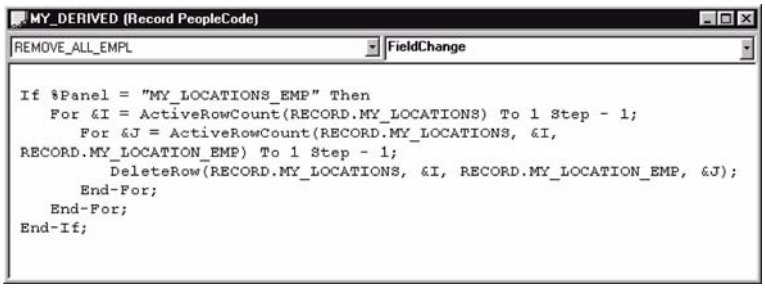


Figure 16.15 DeleteRow at scroll level 2

Example 2

Another push button on the panel in figure 16.10 is labeled "Remove Employees From Location." The code behind this button (figure 16.16) removes employee data from the current scroll level 1 row.

Example 3

An additional button on the panel is labeled "Remove All Locations/Employee." This piece of code removes all locations as well as all the employee data associated with the

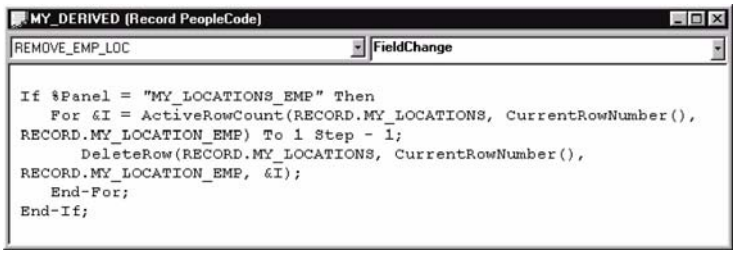


Figure 16.16 DeleteRow at scroll level 2

locations. From a conceptual perspective, it appears that the code to complete this task would be complex. Actually the code (figure 16.17) is not as intricate as one might expect.

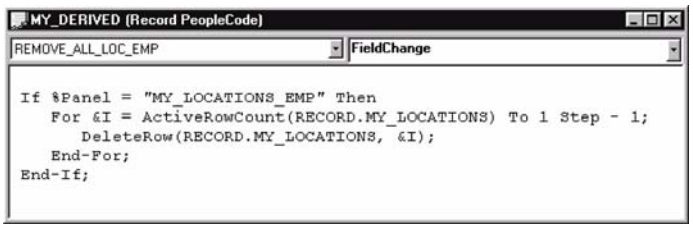


Figure 16.17 Using DeleteRow to remove all rows from both scrolls

The `DeleteRow` function is used in a loop and removes all active records at scroll level 1. Because the record MY_LOCATIONS is parent to the data at scroll level 2, `DeleteRow` removes all lower child level records. The removal of child rows is done automatically by `DeleteRow` as each scroll level 1 record is deleted.

The `ActiveRowCount` function is used to compute the number of active rows. The `DeleteRow` function works from the bottom of the scroll bar and is illustrated by the `Step -1` parameter in the `For` statement.

NOTE `DeleteRow` marks records as deleted. During save processing, rows marked for deletion are removed from the database. `ScrollFlush` clears data from the scroll buffer area only and does not routinely delete rows from the database.

16.3.4 FetchValue

When working with scrolls, it is sometimes necessary to reference data which appears in the individual rows located in the panel scroll buffer. A function used to extract data from these rows is `FetchValue`.

The `FetchValue` function retrieves the value of a field from a row stored in the panel buffer of a scroll area and places it into a variable or fieldname. In addition to

the ScrollPath parameter, FetchValue requires target_row and recordname.fieldname parameters as well. To use FetchValue at the various levels, the following syntax is used:

Level 1

```
FetchValue (RECORD.target_recname, target_row, [recordname.] fieldname);
```

Level 2

```
FetchValue (RECORD.level1_recname, level1_row, RECORD.target_recname,
target_row, [recordname.]fieldname);
```

Level 3

```
FetchValue (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname, target_row, [recordname.]fieldname);
```

Example

Using figure 16.10 let's assume that employee IDs contain a specific level of information based on the type of ID. An ID beginning with the letter "L" represents employee populations excluded from the Operator Class/Location functionality. As a result, the code should prevent any employee ID, which has a leading "L," from appearing on the panel. To accomplish this task, the PeopleCode associated with the "Load All Locations" push button can be rewritten as shown in figure 16.18.

FetchValue is used after the scroll area has been populated using ScrollSelect. In the example, FetchValue is used in a loop that is executed at scroll level 1. FetchValue takes five parameters; MY_LOCATIONS is the level 1

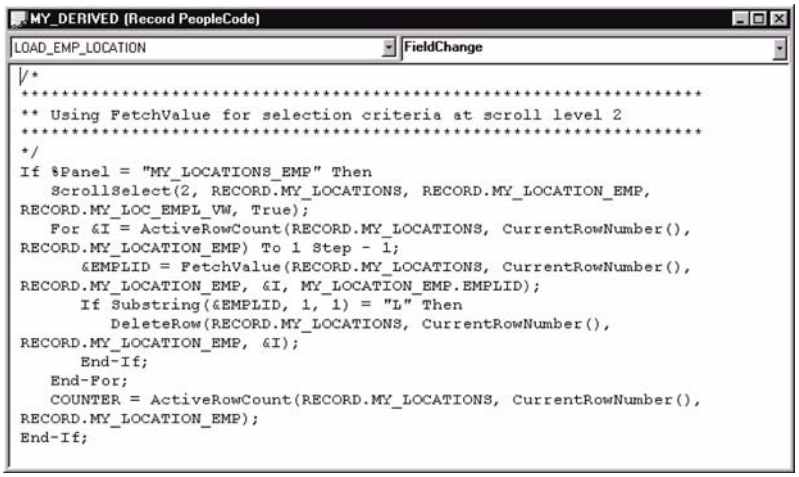


Figure 16.18 Using FetchValue to exclude Emplids

record name. The level 1 row is supplied by the CurrentRowNumber function. The third parameter is the target record name, MY_LOCATION_EMP. The fourth parameter is the variable &I, which contains the row number in the target scroll area. The fifth parameter, [record name.] fieldname is also required. The record name prefix is used because the FetchValue call is made from the MY_DERIVED record, which is different from the record that contains the EMPLID fieldname.

After the call to FetchValue, the variable &EMPLID contains the result and is tested using the SubString function to determine if the first character is an "L." In the example, DeleteRow is used to remove rows that match the selection criteria.

One additional bit of information regarding FetchValue is the empty scroll area. Because the ActiveRowCount function returns 1 when there are no rows in a target scroll, FetchValue may return erroneous data. The PeopleCode in figure 16.18 can be written to include a verification for &EMPLID by adding the following PeopleCode before the If SubString statement.

```
If ActiveRowCount(RECORD.MY_LOCATIONS, CurrentRowNumber(),
   RECORD.MY_LOCATION_EMP) = 1 Then
         If None(&EMPLID) Then
            /*  Scroll is Empty */
            Break;
         End-If;
```

In this example, the verification of an empty scroll is not vital. Other applications using FetchValue, however, may require empty scroll verifications before additional operations are performed.

When working with scrolls, it sometimes becomes necessary to hide rows of data rather than delete or flush them from the scroll area. PeopleCode contains two functions that are used to hide specific rows or an entire scroll area. The functions are HideRow and HideScroll.

16.3.5 HideRow

HideRow is used to hide a specific row and any child rows in subordinate scroll levels. The parameters passed to HideRow are based on the scroll level and target record number where rows are to be hidden. The syntax for HideRow at various levels can be written as follows:

Level 1

```
HideRow (RECORD.target_recname, target_row);
```

Level 2

```
HideRow (RECORD.level1_recname, level1_row, RECORD.target_recname,
target_row);
```

Level 3

```
HideRow (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname, target_row);
```

Example

The example presented in figure 16.18 can be rewritten using `HideRow` in place of the `RowFlush` function. `HideRow` does not remove rows from a scroll. The code shown in figure 16.19 hides the target row ❶, but, during save processing, any hidden rows are written to the database.

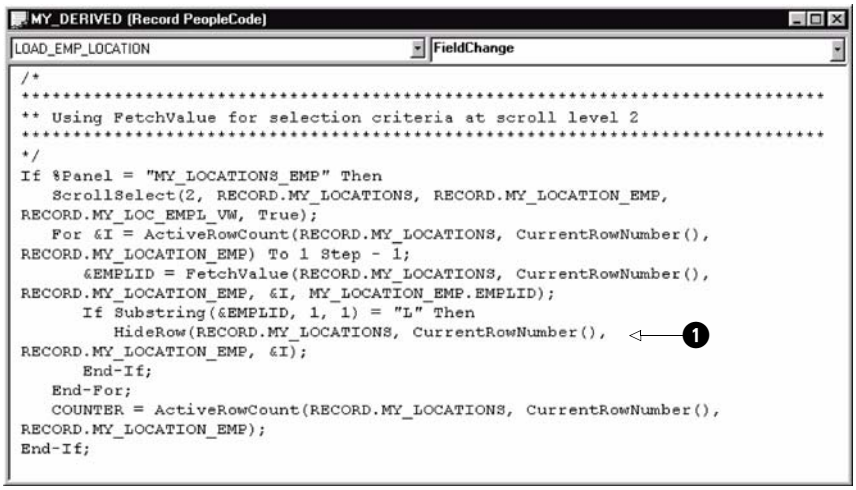


Figure 16.19 HideRow function

In addition to hiding rows of data in a scroll area, it is sometimes necessary to hide an entire scroll bar. Hiding a scroll changes the look of a panel because the scroll bar cannot be viewed. The `HideScroll` function is used to hide a scroll bar and its corresponding data.

16.3.6 HideScroll

`HideScroll` is similar to `HideRow` except that instead of hiding a row, the complete scroll area is hidden, including all data in the scroll and the scroll bar. For each scroll level, the syntax for `HideScroll` can be written as follows:

Level 1

```
HideScroll (RECORD.target_recname);
```

Level 2

```
HideScroll (RECORD.level1_recname, level1_row, RECORD.target_recname);
```

Level 3

```
HideScroll (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname);
```

Example

On some occasions, work scrolls are used to store rows of data subsequently used elsewhere in a routine. Work scrolls do not usually appear on a panel; they can be hidden using `HideScroll`. The objective is to hide the entire scroll for selected locations at level 2. Level 2 contains employee data and is hidden following the `HideScroll`. Figure 16.20 contains the panel without the implementation of `HideScroll`.

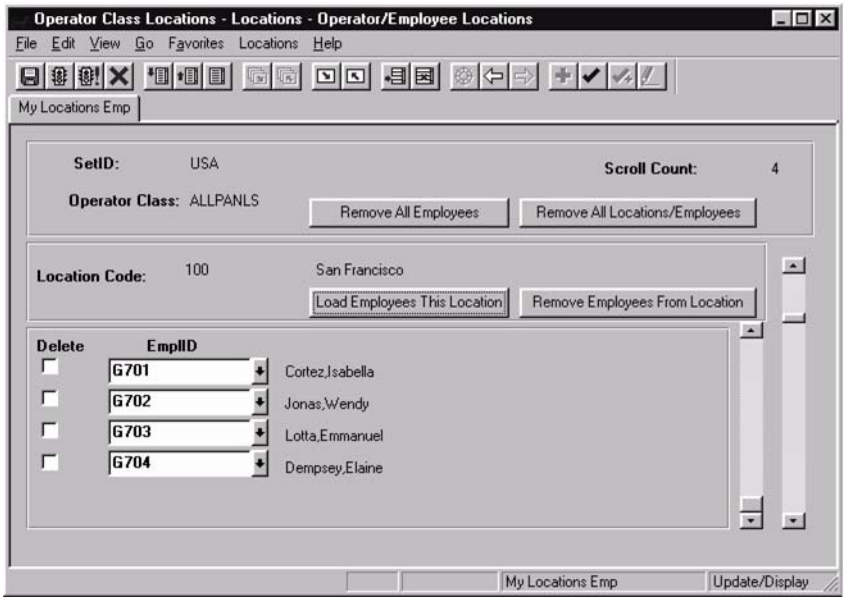


Figure 16.20 Level 2 scroll data

The following example (figure 16.21) contains the necessary PeopleCode required to hide the child scroll area at level 2 for location codes 100 and 300.

The panel illustrating the hidden scroll area for location 100 is shown in figure 16.22. The Delete label that appears on the scroll level 2 area is static text and is not impacted by `HideScroll`.

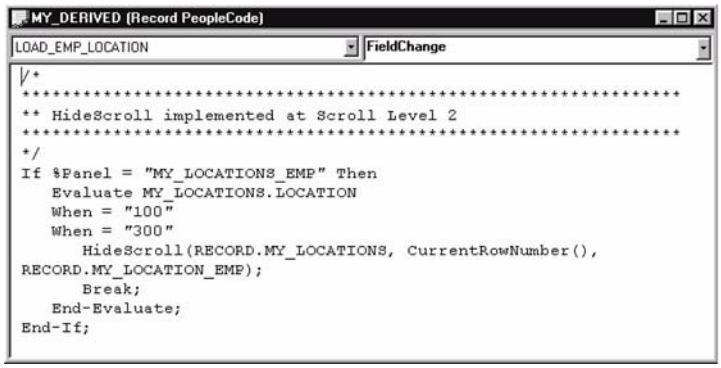


Figure 16.21 Using HideScroll on specific locations

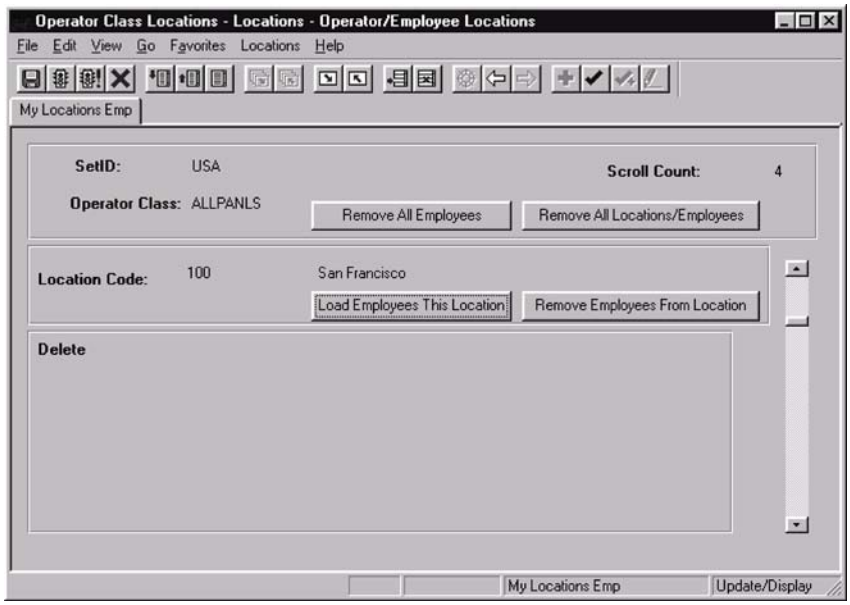


Figure 16.22 Panel after HideScroll is executed

16.3.7 RowScrollSelect

The `ScrollSelect` and `ScrollSelectNew` functions read data from the specified `Select` record into a scroll area and distribute child keys based on their corresponding parent key values. The PeopleCode functions `RowScrollSelect` and `RowScroll-SelectNew` are similar to their counterparts `ScrollSelect` and `ScrollSelect-New`. The difference is that `RowScrollSelect` and `RowScrollSelectNew` do not automatically allocate child rows to their corresponding parent rows.

To illustrate, refer to figure 16.20. When implemented correctly, `ScrollSelect` and `ScrollSelectNew` automatically load child keys `G701`, `G702`, and `G703` on a panel containing two scroll levels when parent key `100` is encountered. Conversely, `RowScrollSelect` and `RowScrollSelectNew` require that the SQL string be used to limit the keys of the rows loaded, to those of the parent row. `RowScrollSelect` requires the specification of the target scroll area, a source record from which to select rows, and the SQL string. The parameters passed to `RowScrollSelect` vary based on the scroll level at which the function is targeted:

Level 1

```
RowScrollSelect (1, RECORD.target_recname, RECORD.sel_recname);
```

Level 2

```
RowScrollSelect (2, RECORD.level1_recname, level1_row,
RECORD.target_recname, RECORD.sel_recname);
```

Level 3

```
RowScrollSelect (3, RECORD.level1_recname, level1_row,
RECORD.level2_recname, level2_row,  RECORD.target_recname,
RECORD.sel_recname);
```

Example

The following code (figure 16.23) demonstrates the use of `RowScrollSelect` at the level 2 scroll. It operates on the panel illustrated by figure 16.10.

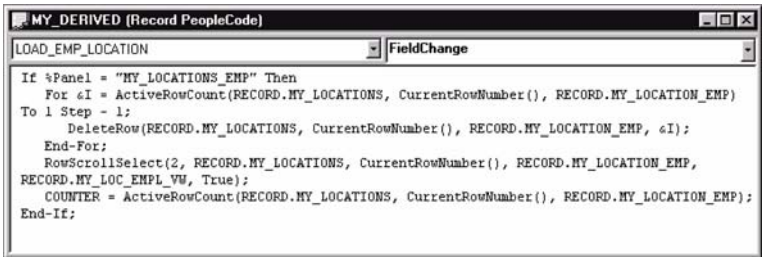


Figure 16.23 RowScrollSelect at level 2

The populated panel using `RowScrollSelect` is shown in figure 16.24.

The example has one flaw. The panel illustrated in figure 16.24 contains more data than the number of employees for the location at level 1. Note the scroll count field and compare it to figure 16.20. A big difference! In the example, the code erroneously does not contain a `WHERE` statement and results in all level 2 rows being loaded into the scroll level 2 buffer, for ALL employees! Without limiting the scroll

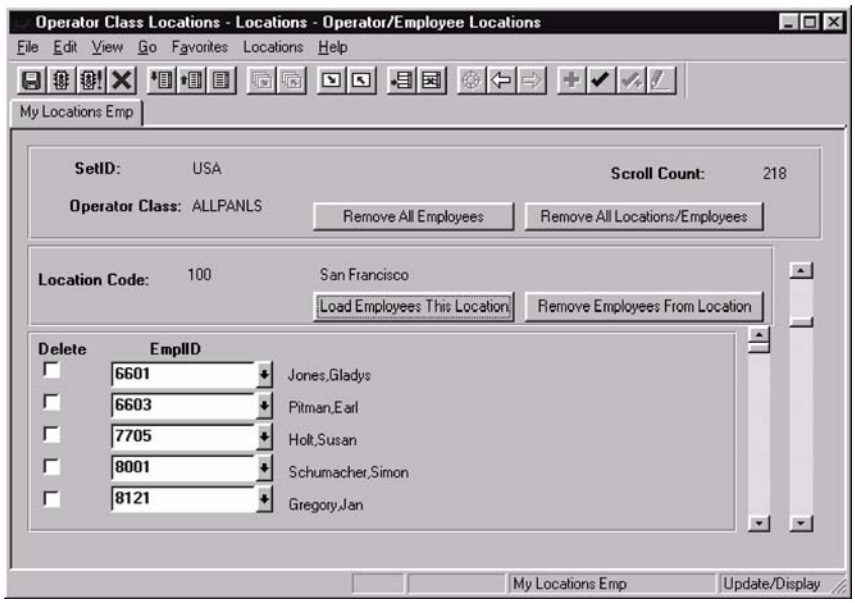


Figure 16.24 Panel using RowScrollSelect

level 2 rows to those of the parent at level 1, data for all employees in the `sel_recordname` are loaded into the scroll buffer.

The correctly coded `RowScrollSelect` function is illustrated in figure 16.25. The `WHERE` SQL clause is inserted to limit keys to those of the parent, ❶. The code produces the same results as those in figure 16.20.

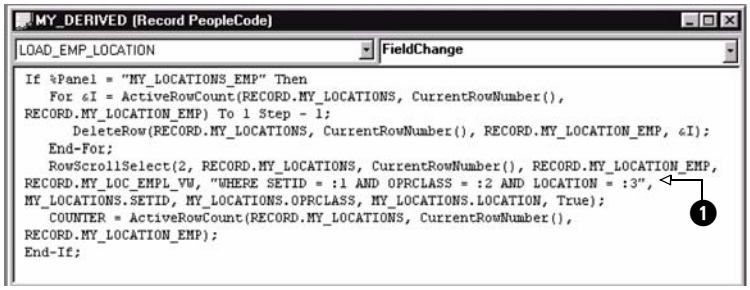


Figure 16.25 Correctly coded RowScrollSelect

A review of the PeopleCode in the example illustrates how `RowScrollSelect` is used with the other PeopleCode statements. First, the panel is verified so that the statements within the scope of the `If` statement are executed when the panel name is

MY_LOCATIONS_EMP. This is necessary because the code is executed from a Derived/Work record, which may contain code used on different panels. The DeleteRow statement is used to delete rows from the scroll and database table. In this example, DeleteRow is used in the context of a loop. Rows are processed from high to low, because rows are renumbered each time they are deleted. The level number identifies the target scroll area is at level 2. The level1_recname parameter is required because the target record is a level 2. CurrentRowNumber is used to obtain the level 1 row. Level 1 row is required when the target record is at level 2. The target recordname is MY_LOCATIONS_EMP and represents the target record into which data will be selected. The sel_recordname parameter is identified by MY_LOC_EMPL_VW, which is a view used to extract the most current effective-dated JOB rows and join them with the corresponding location table entry. This parameter can be the same as the target record name, but, in the example, we are selecting from a view and loading the selected fields into the MY_LOCATIONS_EMP target record. The COUNTER field contains the number of active rows in the scroll area. This count is reflected on the panel.

TIP Understanding the record key definitions for parent and child records facilitates the construction of SQL strings for RowScrollSelect and RowScrollSelectNew functions.

16.3.8 RowScrollSelectNew

RowScrollSelectNew resembles RowScrollSelect the only exception being that RowScrollSelectNew marks records as NEW when they are loaded into the scroll area. RowScrollSelectNew does not automatically place child rows under the corresponding parent data within the scroll buffer. It requires that the SQL string be used to limit the rows loaded into the scroll to those of the parent row. RowScrollSelectNew requires the specification of the target scroll area, a source record from which to select rows and an SQL string. The parameters passed to RowScrollSelectNew vary based on the scroll level at which the function is targeted:

Level 1

```
RowScrollSelectNew (1, RECORD.target_recname, RECORD.sel_recname);
```

Level 2

```
RowScrollSelectNew (2, RECORD.level1_recname, level1_row,
RECORD.target_recname, RECORD.sel_recname);
```

Level 3

```
RowScrollSelectNew (3, RECORD.level1_recname, level1_row,
RECORD.level2_recname, level2_row,  RECORD.target_recname,
RECORD.sel_recname);
```

Example

The PeopleCode example for `RowScrollSelectNew` (figure 16.26) loads the selected data into MY_LOCATIONS using `MY_LOC_OPR_VW` as the sel_recordname. In the example, sel_recordname is a view used to select the most current effective dated LOCATION_TBL entry for the specified `SETID` field. Because the records selected into the target scroll are marked as `NEW`, they are inserted into the database during save processing. For this example, another `SETID` value has been selected. The panel is shown in figure 16.27.

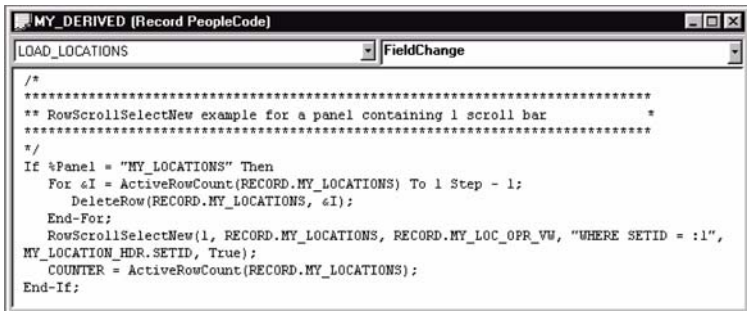


Figure 16.26 RowScrollSelectNew at the level 1 scroll

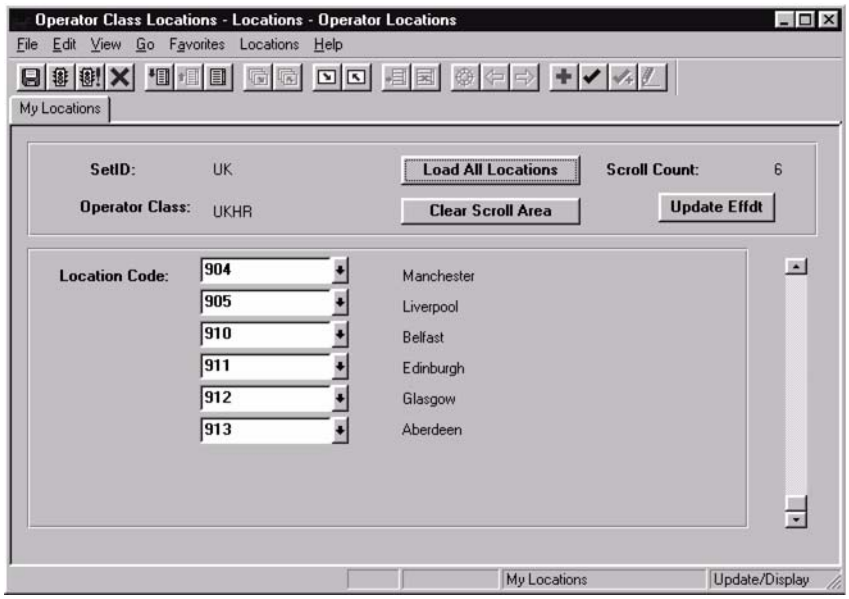


Figure 16.27 Operator/Class locations using RowScrollSelectNew

16.3.9 RowFlush

RowFlush is a scroll function used to remove a particular row of data from the panel scroll and scroll buffer area. RowFlush does not delete rows from the database. RowFlush requires the specification of the target ScrollPath and the target row. The parameters passed to RowFlush vary based on the scroll level at which the function is targeted. To use RowFlush on specific levels, it can be coded as follows:

Level 1

```
RowFlush (RECORD.target_recname, target_row);
```

Level 2

```
RowFlush (RECORD.level1_recname, level1_row, RECORD.target_recname,
target_row);
```

Level 3

```
RowFlush (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname, target_row);
```

Example

The level 2 scroll area in figure 16.24 contains a checkbox labeled Delete. RowFlush can be used after the push button labeled "Load Employees This Location" has been activated. The associated employee data are selected into the scroll area and can subsequently be saved. If the need to delete employee data are required, the Delete Row (F8) toolbar option can be used. The F8 or Delete Row however requires that we confirm the delete, thereby adding an additional step for the user. The Delete checkbox can be applied during save processing to use RowFlush, which removes the identified rows prior to their insertion into the database table. Figure 16.28 illustrates the use of the panel before it is saved.

The illustration identifies two rows that have the Delete checkbox indicator turned on. During save processing, the code shown in figure 16.29 is executed. For rows having the DELETE_ROW field set to "Y", the PeopleCode calls the RowFlush function. The result of the RowFlush function is illustrated in figure 16.30.

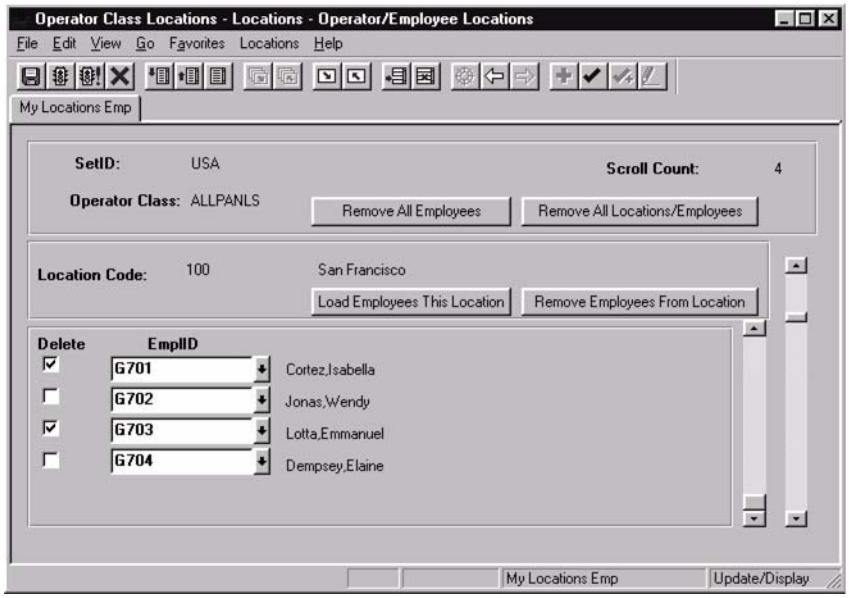


Figure 16.28 Using RowFlush to remove scroll data

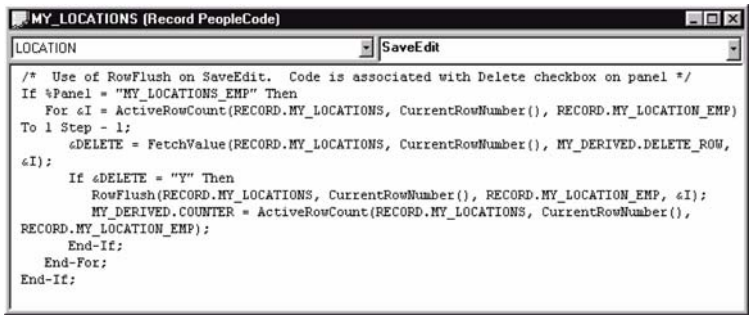


Figure 16.29 RowFlush PeopleCode at level 2

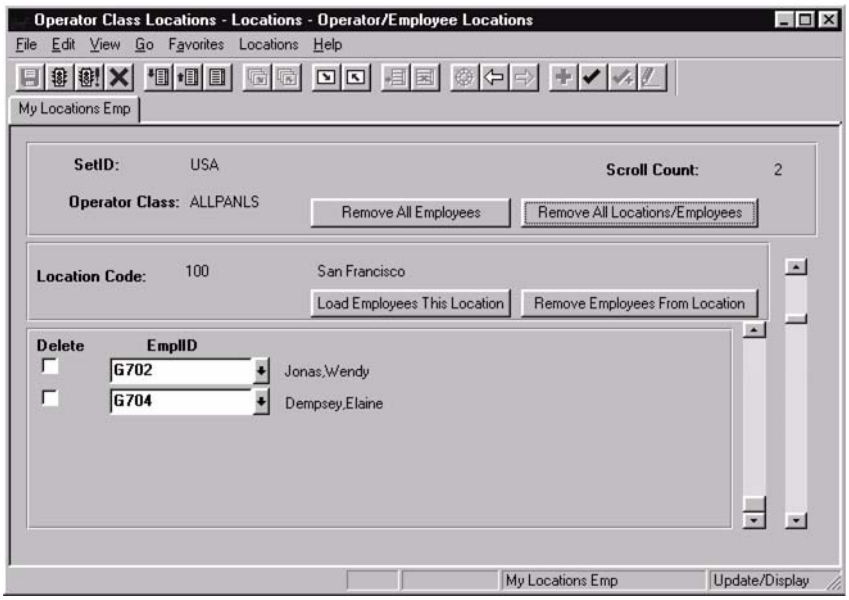


Figure 16.30 Resulting panel with selected rows removed

The PeopleCode surrounding `RowFlush` in figure 16.29 contains verification of the panel name MY_LOCATIONS_EMP. In the example, the code is executed from the `SaveEdit` event of the record MY_LOCATIONS. `FetchValue` is used to retrieve the indicator field MY_DERIVED.DELETE_ROW. The field is checked for a value `"Y"`. A `True` condition causes the `RowFlush` function to be called. The `RowFlush` parameters include level1_recname. This parameter is required because the target record is at level 2. The parameter is specified as `RECORD.MY_LOCATIONS`. Because the target record is at level 2, the level1_row parameter is also required. `CurrentRowNumber` is used to obtain the level 1 row. The target record name is specified as `MY_LOCATIONS_EMP`. At level 2 the last parameter passed to `RowFlush` is `target_row`. The variable `&I`, which is used to loop through the data at scroll level 2, identifies the row number to be removed from the specified target scroll area. An additional line of code references MY_DERIVED.COUNTER, which is updated after `RowFlush` so that the panel contains the actual number of active rows following save processing.

It is important to note that `RowFlush` does not remove rows from the database; it only removes them from the panel scroll buffer. In the example presented, `RowFlush` does not work for data that has been saved to the database, then subsequently deleted using the Delete indicator. Let's review again. Refer to figure 16.20 and assume the data on that panel has now been saved. The panel is then subsequently retrieved, and two rows are checked off (figure 16.28) with the assumption that the

rows will be deleted during save processing. Following save processing, the rows marked for deletion will be gone and will appear to have been deleted. However, when the scroll buffer is reloaded with data from the table or view, the "deleted" rows reappear. For this reason, `RowFlush` is limited to specific applications. It is more common to use `DeleteRow` because data are removed from the scroll and deleted from the database.

Figure 16.31 illustrates the `DeleteRow` function as a replacement for `RowFlush`.

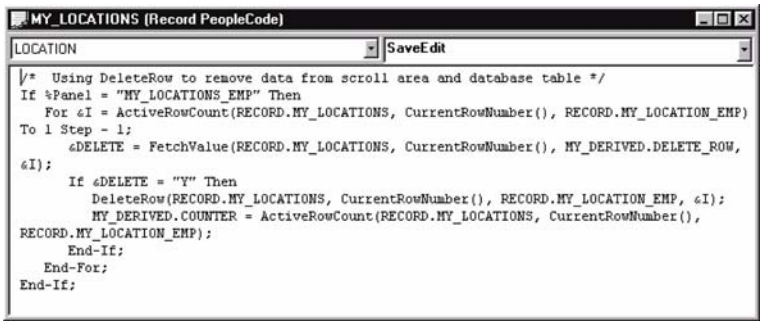


Figure 16.31 Using DeleteRow in place of RowFlush

TIP The `RowFlush` example does not necessarily have to be replaced with `DeleteRow` to obtain the desired functionality. The use of the Delete Row toolbar icon or F8 also removes rows from the scroll area while maintaining the `RowFlush` PeopleCode. This will require an additional step for the user, however.

16.3.10 UpdateValue

The `UpdateValue` function works in a similar manner to `FetchValue` to update the value of a field using the value parameter passed to the function. `UpdateValue` requires the `target_row` and `recordname.fieldname` parameters as well as a value that can be specified as a variable, constant, or record field. To use `UpdateValue` at various levels, the following syntax is used:

Level 1

```
UpdateValue (RECORD.target_recname, target_row, [recordname.] fieldname,
value);
```

Level 2

```
UpdateValue (RECORD.level1_recname, level1_row, RECORD.target_recname,
target_row, [recordname.]fieldname, value);
```

Level 3

```
UpdateValue (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname, target_row, [recordname.]fieldname,
value);
```

Example

The Operator Class/Locations panel contains a button labeled "Update Effdt." When the button is activated, a secondary panel is displayed which accepts a date value. The date entered from the secondary panel is used to update an effective date field located on MY_LOCATIONS record. The effective date is used for reporting and internal tracking purposes, but does not appear on the panels illustrated. Figure 16.32 illustrates the secondary panel.

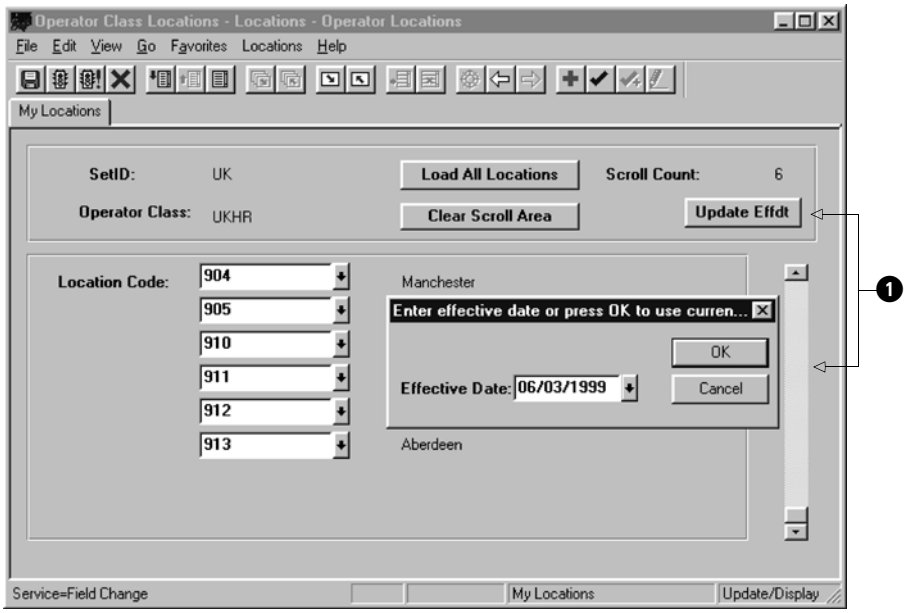


Figure 16.32 Update Effdt secondary panel associated with UpdateValue

The PeopleCode utilizing `UpdateValue` is shown in figure 16.33. This code works in conjunction with the Update Effdt push button (❶ in figure 16.32) and uses the date entered from the secondary panel.

The `UpdateValue` parameters specify target record name as `RECORD.MY_LOCATIONS`. The target_row parameter is specified as a variable `&I`, which is incremented using the `For` statement. The variable `&I` represents the row numbers in the target scroll area. Record name is required as a prefix to fieldname

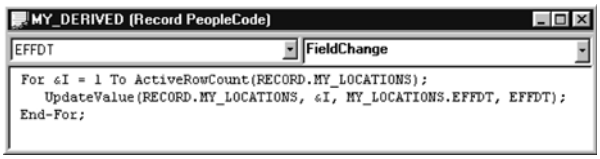


Figure 16.33
UpdateValue PeopleCode

because the PeopleCode is executed from the MY_DERIVED record. As a result, the target fieldname is prefixed with MY_LOCATIONS. The last parameter received by `UpdateValue` is a value. In the example, the value is entered into the secondary panel and contained in the work field MY_DERIVED.EFFDT.

TIP Regular data assignment can be used instead of `UpdateValue` when the field to be updated appears on the same record as the PeopleCode.

16.3.11 TotalRowCount

As previously discussed, it is sometimes necessary to identify the number of rows in a scroll area. `ActiveRowCount` is a function used to count the number of active rows in a target scroll area. When it is necessary to count active as well as deleted rows, the `TotalRowCount` function can be used. `TotalRowCount` returns the aggregate number of rows in a scroll area including deleted rows.

The parameters required by `TotalRowCount` are based on the target scroll level for which a count is required:

Level 1

```
TotalRowCount (RECORD.target_recname);
```

Level 2

```
TotalRowCount (RECORD.level1_recname, level1_row, RECORD.target_recname);
```

Level 3

```
TotalRowCount (RECORD.level1_recname, level1_row, RECORD.level2_recname,
level2_row,  RECORD.target_recname);
```

Example 1

To obtain the total number of rows contained in the scroll area of the Operator Class/ Location panel, the following code using `TotalRowCount` can be used:

```
MY_DERIVED.COUNTER = TotalRowCount(RECORD.MY_LOCATIONS);
```

Example 2

The following example uses `TotalRowCount` on the level 2 scroll to count the number of child rows containing employee data:

```
MY_DERIVED.COUNTER = TotalRowCount (RECORD.MY_LOCATIONS,
CurrentRowNumber(), RECORD.MY_LOCATION_EMP);
```

TIP Rows marked as "deleted" remain in the buffer until an SQL commit is issued after the `SavePostChg` PeopleCode event.

KEY POINTS

1. A panel can contain up to three scroll levels. A panel with two scroll levels contains primary records at occurs level zero, occurs level 1, and occurs level 2.
2. The level zero record is considered the parent row and level 1 is the child. A level 1 row is parent to a level 2 row. Each level can contain multiple records, but only one primary record can exist for each level specified.
3. During processing of data buffers, records at occurs level zero are handled before occurs level 1. A panel containing two occurs levels will process the level zero records first, then a single row of level 1 data, and all level 2 rows which are children of the level 1 row.
4. Some PeopleCode functions used to operate on scrolls include `ScrollSelect`, `ScrollSelectNew`, and `ScrollFlush`. Additional functions such as `RowSelect`, `RowSelectNew`, and `RowFlush` require the use of the `WHERE` block in the SQL string to match parent keys.
5. Other functions used to complement scroll processing include `FetchValue`, `UpdateValue`, `ActiveRowCount`, and `CurrentRowNumber`. These functions allow us to process specific data with a scroll area.

CHAPTER 17

Function libraries

A Function Library is a collection of one or more routines that can be called from another program. Function Libraries offer us the opportunity to reuse code and to write special routines which can be shared by applications running under PeopleTools. In addition to writing and calling functions written in the PeopleCode language, we can also call external programs written in languages such as C/C++. A function usually accepts one or more values and can either return a value or not.

17.1 FUNCTION OVERVIEW

Functions are pieces of code that, in their most basic form, accept a string of parameters and can either return a value, return no value or in some instances call another function and eventually return home. Functions are everywhere and not only in PeopleCode. An SQL `Select` statement that utilizes specialized date functions will enable formats to be used as input and output based on patterns passed to the functions. Similarly, the SQR statement `RTRIM` is also a function. COBOL has functions such as `NUMVAL`, which work in conjunction with a `Move` or `Compute` statement. All types of functions share these characteristics:

- They can be called from almost anywhere in a program.
- They may be called once or thousands of times by a program.
- They perform a specific task that can be shared by systems. Some functions are bundled into software packages and are used to make the software more efficient or to relieve the application developer from writing redundant routines.
- They do not have to be written in the same language as the calling routine. PeopleCode can call C functions as well as C++. If we are daring enough, we can also write callable Assembler routines.

An important requirement of a function is that it establishes a calling convention to allow parameters to be passed back and forth. A function which formats dates could be written to accept one format and pass back another format. The function must know what the input format is and what the output format will be. Assume we have a function called `DATE_FUNCTION`, which accepts a Julian date in the form `YYDDD` and returns the value `YYYYMMDD`. Such a function will be flawed in several ways. First, the function should be flexible enough that it can accept a variety of formats and output them in the manner desired by the calling program. So while we may have some issues with a `YYDDD` format we could also accept `D/M/Y`, `MMDDYY`, or `dd-Mon-YYYY` where `Mon` is a three character representation of Month. January 1, 2000 would be `01-JAN-2000`. The routine would also require indicators to identify the input and output formats. This style of coding helps eliminate redundancy and allows for the localization of functions.

NOTE PeopleCode functions cannot be called by any other type of programs written in a language other than PeopleCode.

PeopleCode programs utilize various types of functions. The function types are as follows:

- built-in
- internal
- external PeopleCode
- external non-PeopleCode

17.2 PEOPLECODE BUILT-IN FUNCTIONS

Built-in functions are the standard PeopleCode functions developed by PeopleSoft which can be called without being declared. Built-in functions are used to manipulate dates, strings, scrolls, and messages. Functions can be grouped into functional categories. Built-in functions differ from Internal and External PeopleCode functions because they do not necessarily have to be written in PeopleCode. Many built-in functions are written in C++.

Functions, by definition, are basically routines that can be shared among different programs or, at the very least, among programs common to an application. A good example of such a function is the PeopleCode built-in function `RTrim`. This is a great little built-in function that removes characters, usually blanks, from the rightmost portion of a string. You pass it the string or list of strings you wish to remove from the source string, and it works just fine. Most of the PeopleCode built-in functions can be used by other PeopleCode programs when necessary.

PeopleCode built-in functions can be grouped into functional categories. For the current release of PeopleTools, some of the most frequently used categories and corresponding functions are:

- Conversion
- Date/Time
- Effective Date/Sequence
- Logical
- Mail
- Math
- Message Catalog/Display
- Panel Buffer
- Panel Control
- Process Scheduler
- Save/Cancel
- Scroll Functions
- SQL
- String
- Transfers
- Validation

Many of the frequently used functions in these categories are listed in appendix E. Some have been discussed in previous chapters. For example, `Message` functions are discussed in chapter 14 and the `SQLExec` function is illustrated in chapter 15. In chapter 16, we examined scroll functions and how they can be applied to panels containing multiple scroll levels. A discussion of a few more frequently used categories follows.

17.2.1 Conversion functions

Functions categorized as conversion are primarily used to translate data values from one character set or data type to another. Some functions belong to more than one category. An example of this is the `String` function, which can be used to convert from a non-string data type to a string. The `String` function is categorized as both a `String` and `Conversion` function.

Another example of a conversion function is `Char`. The `Char` function is used to convert numeric values to their corresponding character values. An example of the `Char` function follows:

```
&CHAR_STRING = Char(72) | Char(69) | Char(76) | Char(76) | Char(79);
```

The target variable contains `HELLO` based on the ASCII number values passed to the `Char` function.

17.2.2 Date/Time functions

PeopleCode provides a number of date and time handling functions categorized as Date/Time. Functions in this category can be used to convert dates represented by numbers into a `Date` data type. Date/Time functions can also be used to extract the date or time portion of a `DateTime` data type. An example of a Date/Time function is `Date`. The `Date` function converts a numeric date into a `Date` data type. An example of `Date` follows:

```
&DATE_NUMBER = 20000122;
&DATE_OUT = Date(&DATE_NUMBER);
```

The variable `&DATE_OUT` contains a date represented as 2000-01-22.

The `DateValue` function is another Date/Time function that accepts a date in the Windows regional format setting and returns a `Date` data type. Assuming the Windows regional date setting is `yy/mm/dd`, the `DateValue` function can be applied in this manner:

```
&DATE_FORMAT = "000122";
&THIS_DATE = DateValue(&DATE_FORMAT);
```

After the `DateValue` function is called, the variable `&THIS_DATE` contains a `Date` data type containing 2000-01-22.

`Time` is a function which receives a number representing a time value and obtains a `Time` data type. The parameter passed to the `Time` function is a number based on a 24-hour clock with the format `HHMMSS[.SSSSSS]`.

```
&REPORT_TIME = 143041.000001;
&TIME_VALUE = Time(&REPORT_TIME);
```

The Time function obtains a time value from the numeric variable &REPORT_TIME and stores it in &THIS_TIME, which now contains the value 14.30.41.000001. Notice that the precision goes out to .000001 seconds.

Additional built-in functions that operate on date or time values can be used to perform arithmetic on date or time variables. AddToDate is a function which receives a date parameter and three values representing number of years, months, and days. The values are added to the date value supplied and a Date value is returned which contains the original date and the aggregate value represented by years, months, and days. Here is an example of AddToDate.

```
&RETURN_DATE = AddToDate(%Date, 5, - 4, 3);
```

Based on the preceding example, when the AddToDate function receives control, the function receives the current system date, the number of years represented as 5, number of months as –4, and number of days as 3. Assuming the current date is 2000-01-22, the value of &RETURN_DATE is now 2004-09-26. This represents a date five years in the future, less four months, plus three days from the current date.

NOTE Passing negative numbers to represent years, months, or days has the effect of subtracting from the date value passed.

17.2.3 Effective Date/Sequence functions

In PeopleSoft, most applications are built around the concept of Effective Date and Effective Sequence. With effective-dated records, data can be managed chronologically in the order of events. Under varying circumstances, the need for multiple records with the same effective date is inevitable. To handle such conditions, the Effective sequence key field is utilized. The combination of Effective Date (EFFDT) and Effective Sequence (EFFSEQ) enables the existence of unique rows with the same effective date. Functions categorized as Effective Date/Sequence operate primarily on scroll areas containing effective-dated rows or tables which contain Effective Date and Effective Sequence as part of their key structure.

The CurrEffdt function is used to return the effective date of the current record on the specified scroll level. The function can be used to extract the effective date, which is returned as a Date value or used in a conditional statement.

To extract the effective date of the current record we can code:

```
&RETURN_DATE = CurrEffDt(CurrentLevelNumber());
```

The function can also be used in a conditional context. For example, when specific processes are performed based on effective date values, the following can be used:

```
If CurrEffDt(CurrentLevelNumber()) < %Date Then
   Audit_Changes(EMPLID);
End-If;
```

In this example, the Audit_Changes function is executed when the effective date of the current record is less than the %Date system variable, which stores the current system date.

The CurrEffSeq function is used to obtain the effective sequence of a specified scroll level. The following example grays out the Account Code field on a panel when the current Effective Sequence contains a higher value than the prior record:

```
If CurrEffSeq(1) > PriorEffdt(EFFSEQ) Then
   Gray(ACCT_CD);
End-If;
```

Based on data in table 17.1, let's assume that the HRMS application contains a current record with an effective date of 1999-07-01 and an ACTION_REASON of XFR. When the ACTION_REASON value for the next record is required, the NextEffDt function can be used. NextEffDt returns the value of the specified record field which exists in the next effective-dated row. To obtain the next ACTION_REASON, the function call can be written as:

```
&NEXT_ACTION_REASON = NextEffdt(ACTION_REASON);
```

After the function call, the variable &NEXT_ACTION_REASON contains PRO. Using table 17.1, when the current record's Effective Date is 2000-01-05, the NextEffDt function call is skipped because a next record does not exist.

Another function categorized as Effective Date/Sequence is PriorEffDt. This function works in contrast to NextEffDt and is used to return the contents of the specified field from the prior effective-dated row. Using table 17.1, the following statement retrieves the XFR ACTION_REASON when the current effective-dated row is 2000-01-05.

```
&PRIOR_ACTION_REASON =PriorEffdt(ACTION_REASON);
```

A statement using PriorEffdt is ignored when the current effective-dated row is 1996-01-01.

Table 17.1 Effective Date and Action/Reasons

EFFDT	ACTION_REASON
2000-01-05	PRO
1999-07-01	XFR
1998-06-30	MER
1996-01-01	HIR

NOTE `CurrEffdt` retrieves the value of the current row's effective date. `NextEffDt` and `PriorEffDt` are used to obtain the value of a specified field that does not necessarily have to be an effective date.

17.2.4 Logic functions

Functions in this category are used to test for the existence of blank values. The commonly used All function is used to determine whether one or more fields contain a value. The `All` function statement is useful in the `SaveEdit` PeopleCode event if we wish to verify that one or more fields have been entered. For the Problem Tracking application, the `All` function can be used to determine if specific fields have been entered. The `All` function can be used to test several fields with one call. The following code can be placed into the `SaveEdit` event for one of the specified fields; it verifies that the three fields passed as parameters each contain a value before calling `MyScheduleFunction` function:

```
If All(PRIORITY, MY_USER_ID, MY_PROBLEM_TRACKER) Then
   MyScheduleFunction();
End-If;
```

An additional logic function used to test for the existence of values is `None`. The function returns `TRUE` if the field or list of fields supplied do not contain a value. A Boolean `FALSE` is returned when one or more fields contain a value. A variation on the `All` function can be written using `None` to test for the existence of specific values.

```
If None(PRIORITY, MY_USER_ID, MY_PROBLEM_TRACKER) Then
   Error ("Enter all required fields");
End-If;
```

PeopleCode programs can be reduced in terms of lines of code by combining multiple fields when using Logic functions such as `All` or `None`.

17.2.5 Math functions

Functions categorized as Math operate primarily on numeric data and can be used to assist with complex calculations. Some of these functions are invaluable when using PeopleSoft Financials, Payroll, or Manufacturing applications. When assigning numeric data elements that result from a multiplication or division operation, it is sometimes necessary to obtain a specific number of decimal positions before inserting values into a database table. One function used to accomplish this task is `Round`. The `Round` function returns a decimal number rounded up to the number of positions

specified by the second parameter. If we are required to round the HRMS data element COMPRATE to three decimal positions, the following statement can be used:

```
JOB.COMPRATE = Round(JOB.COMPRATE, 3);
```

An additional function, which operates on numeric data but is not interested in decimal positions, is the `Int` function. The `Int` function removes decimal positions from a number and returns an integer value. As an example let's assume one of the payroll subroutines calculates hours worked, based on whole hours only. The following code can be used to calculate overtime hours using the `Int` function:

```
If &HOURS_WORKED > JOB.STD_HOURS Then
   &OVERTIME_HOURS = Int(&HOURS_WORKED) - JOB.STD_HOURS;
End-If;
```

Based on that `Int` example, employees working overtime hours that are not whole hours will not be too happy on payday. That is because the `Int` function does not perform any rounding. In the example, when the value of `&HOURS_WORKED` is `65.753` the `Int` function uses `65`.

When dividing numbers, it is sometimes necessary to interpret the value of the remainder field. This can be accomplished using the `Mod` function. `Mod` divides one number by another and returns a value representing the remainder. We can see an application of the `Mod` function when a specified number is divided by number of years. Let's assume we have shares of stock and wish to allocate them evenly in whole numbers over a period of seven years. One rule is that, when the number cannot be divided evenly, the first year contains one additional share. This may appear complicated at first but can be facilitated using `Mod`:

```
&YEARLY_STOCK = (&STOCK_SHARES / 7);
&YEARLY_STOCK = Int(&YEARLY_STOCK);
&REMAINING_SHARES = Mod(&STOCK_SHARES, 7);
If &REMAINING_SHARES <> 0 Then
   &FIRST_YEAR_STOCK = &YEARLY_STOCK + 1;
End-If;
```

Observe how the routine uses both `Mod` and `Int` to determine the yearly stock and any remaining shares.

17.2.6 Panel buffer functions

Functions in this category are used to identify changes to records residing in panel buffers. A panel buffer can contain scroll data as well as non-scroll data. Panel Buffer functions can also be used to mimic operator functionality such as `InsertRow` (F7) and `DeleteRow` (F8). These functions are essential during save processing because they can be used to perform specific routines based on user actions.

The Problem Tracking application referred to throughout this book uses a main data entry panel (figure 17.1) which contains several editable fields. The task of identifying changes made to fields can be accomplished by including PeopleCode which tests each record field for changes. Such a task requires several lines of code. A more efficient method can be applied by incorporating the `RecordChanged` panel buffer function. The function returns a Boolean (`TRUE`) when the contents of a record have been changed on a panel or modified programmatically since being retrieved from the database.

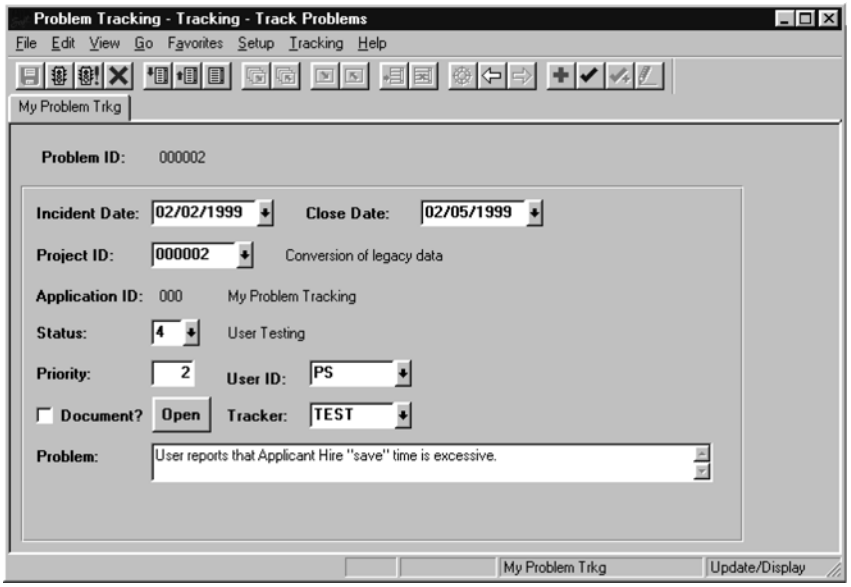


Figure 17.1 Problem Tracking panel

An example of `RecordChanged` can be applied as follows:

```
If RecordChanged(RECORD.MY_PROBLEM_TRKG) Then
   &RETURN_VALUE = MyScheduleFunction();
End-If;
```

`FieldChanged` is another panel buffer function. However, unlike `RecordChanged` which operates on the entire record, `FieldChanged` can be used to identify if one or more fields have been modified The function thereby operates at the field level. To test specific fields in Problem Tracking, a statement can be written as illustrated by the following example:

```
If FieldChanged(MY_PROBLEM_TRKG.PRIORITY) Or
      FieldChanged(MY_PROBLEM_TRKG.MY_USER_ID) Or
```

```
      FieldChanged(MY_PROBLEM_TRKG.MY_PROBLEM_STATUS) Or
      FieldChanged(MY_PROBLEM_TRKG.MY_PROBLEM_TRACKER) Then
   &RETURN_VALUE = MyScheduleFunction();
End-If;
```

Additional panel buffer functions can be used to delete records or identify records marked for deletion. When it is necessary to delete a record, the `DeleteRecord` panel buffer function can be used. The function deletes a level zero record and any associated child rows. The records are marked for deletion and are removed from the database during save processing. The function accepts any field values from the target record to be marked for deletion:

```
If &DELETE = "Y" Then
   DeleteRecord(MY_LOCATIONS.OPRCLASS)
End-If;
```

This example deletes the corresponding record from the table MY_LOCATIONS as well as any associated child rows.

Records marked for deletion by a panel buffer function such as `DeleteRecord` can be identified using the `RecordDeleted` function. An example of how `RecordDeleted` can be applied to scroll data at level 2 follows:

```
If RecordDeleted(RECORD.MY_LOCATIONS, CurrentRowNumber(),
RECORD.MY_LOCATION_EMP) Then
   &RETURN = My_Audit_Function();
End-If;
```

Panel buffer functions can also be used to insert records into a scroll or to identify records as new to the database. The `InsertRow` function is used to insert a new row of data into the scroll buffer. The operation is followed by the `RowInsert` PeopleCode event. `InsertRow` mimics the F7 operator function. To insert data at the level 2 scroll, `InsertRow` can be written in the following manner:

```
InsertRow(RECORD.MY_LOCATIONS, CurrentRowNumber(), RECORD.MY_LOCATION_EMP);
```

During save processing, records added to the database can be identified using the `RecordNew` panel buffer function. When used in `SaveEdit` before the record is inserted into the database, `RecordNew` is used to identify new records. This can be useful when special routines are necessary for newly added data. It is important to identify new data which can be added using F7 (Row Insert) or the `InsertRow` function. In order to recognize a new record at scroll level 1 for the table MY_LOCATIONS, the following code can be used:

```
If RecordNew(RECORD.MY_LOCATIONS) Then
   &RETURN = My_Audit_Function();
End-If;
```

17.2.7 Panel control functions

Consider the following scenario: we have implemented the HRMS module among a large number of users, many having access to the same panels. There are, however, some clients who do not require access to a number of fields distributed throughout the panel group. The objective is to allow access to these panels without displaying the fields. Several methods exist for accomplishing this task. One method involves creating several panels without the specified fields. We then have to create a new panel group and add the corresponding panel group to a menu. This approach requires additional panels. An alternative and more efficient method is to use PeopleCode. Using system variables and specific PeopleCode panel control functions, the selected fields can be hidden or made inaccessible on selected panels. One panel control function that can be utilized is the `Hide` function. `Hide` can render the specified panel fields invisible. Assuming the panels containing these fields are on a specific panel group, the following code can be applied to accomplish this task:

```
If (Substring(%PanelGroup, 1, 8) = "SPECIAL_DATA") Then
   Hide(COMPRATE);
   Hide(SAL_ADMIN_PLAN);
   Hide(GRADE);
   Hide(HOURLY_RT);
   Hide(ANNUAL_RT);
   Hide(MONTHLY_RT);
End-If;
```

Other circumstances exist in which hiding fields in not necessary. If we simply wish to prevent certain editable fields from being written over, the panel control function, `Gray` can be used. The `Gray` function is commonly used in the `RowInit` event and can also be used in events such as `FieldChange` after the contents of a field are modified.

In the Problem Tracking application, when an issue is considered closed, certain fields can be presented as display only using the `Gray` built-in function contained in the following code:

```
If MY_PROBLEM_STATUS = "5" Then
   Gray(PRIORITY);
   Gray(MY_USER_ID);
   Gray(MY_PROBLEM_TRACKER);
   Gray(DESCRLONG);
End-If;
```

Additional Panel Control functions can be used to make panel fields visible again. The `UnHide` function can be used for this task, but only works on fields that were hidden using the `Hide` function. Fields initially set to invisible, based on the panel field properties tab, are not impacted by `UnHide`. The following example is used to `Hide` or `UnHide` a field on the Problem Tracking panel:

```
If MY_PROBLEM_STATUS <> "5" Then
   Hide(MY_PROBLEM_RESOLTN);
```

```
Else
   UnHide(MY_PROBLEM_RESOLTN);
End-If;
```

The UnGray function can be used to make a previously protected field editable using Gray. UnGray is commonly used in the RowInit event and can also appear in events such as FieldChange after the contents of a field are modified.

In the Problem Tracking application we can see the UnGray function when a Problem Status is changed from resolved back to another status:

```
If PriorValue(MY_PROBLEM_STATUS) = "5" And
      MY_PROBLEM_STATUS <> "5" Then
   UnGray(PRIORITY);
   UnGray(MY_USER_ID);
   UnGray(MY_PROBLEM_TRACKER);
   UnGray(DESCRLONG);
End-If;
```

17.2.8 Save/Cancel functions

Functions in the Save/Cancel category enable the PeopleCode program to force a cancel from an active panel or to save the contents of a panel. The DoCancel function is used to cancel the activity on a panel and imitate the Esc or Cancel toolbar. Figure 17.2 illustrates the use of DoCancel after a Warning statement has been issued. The panel is canceled after the user replies OK to the warning message.

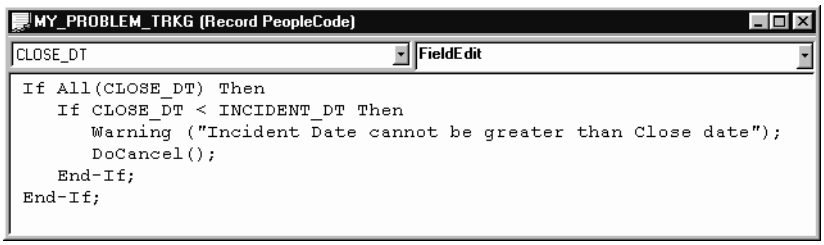


Figure 17.2 Using the DoCancel function

NOTE If Error were used in place of Warning, the DoCancel function would have no effect, because the Error function negates any further processing until the error has been corrected.

In addition to providing the ability to cancel a panel, PeopleCode functions enable save processing to be performed without a user save action. This can be accomplished using the DoSave function which can be executed from specific PeopleCode events. DoSave performs save processing at the conclusion of the current PeopleCode program

in the FieldEdit, FieldChange, and MenuItemSelect events. Any statements following DoSave are executed through the remainder of the program and then the associated save events are triggered. These events include SaveEdit, SavePreChg, WorkFlow, and SavePostChg. An example of DoSave in the Problem Tracking application is illustrated by figure 17.3.

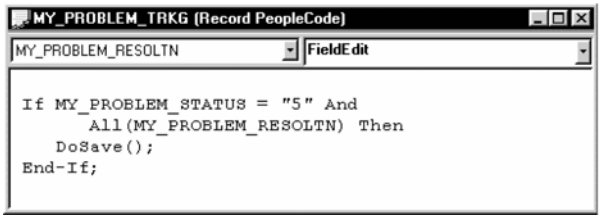


Figure 17.3
Using DoSave function

17.2.9 String functions

String functions enable the developer to manipulate String data types. This category of functions can be used to extract portions of a string, determine the length of a string, or to remove specified leading and trailing characters.

The Substring function references a string for a specified length and starting position. The following example uses Substring to verify a portion of the %PanelGroup system variable and to hide a panel field:

```
If (Substring(%PanelGroup, 1, 8) = "SPECIAL_DATA") Then
   Hide(COMPRATE);
End-If;
```

LTrim and RTrim are functions that can be used to remove specified leading or trailing characters respectively. Both functions accept up to two parameters. When the second parameter is specified, any characters appearing in the parameter string are removed from the first string parameter. If the second parameter is not supplied, any leading or trailing blanks are removed by default. MY_PROBLEM_RESOLTN is a field that contains a text field. To remove any unwanted trailing asterisks from this field, RTrim can be applied as follows:

```
MY_PROBLEM_RESOLTN = RTrim(MY_PROBLEM_RESOLTN, "*");
```

It sometimes becomes necessary to represent data in one common format or case. This may be necessary when data are maintained for interface purposes and are sent to a third party application. Some of these third party applications may require that alphabetic data are sent in a specific case only. PeopleCode functions Lower and Upper can be used to convert alphabetic characters to lower case or upper case, respectively.

The following example illustrates how the field MY_PROBLEM_RESOLTN can be converted to uppercase:

```
MY_PROBLEM_RESOLTN = Upper(MY_PROBLEM_RESOLTN);
```

Alternatively, we can convert the text in the field MY_PROBLEM_RESOLTN to all lowercase using the `Lower` function:

```
MY_PROBLEM_RESOLTN = Lower(MY_PROBLEM_RESOLTN);
```

Another application of the `Upper` function can be illustrated when comparing two fields which may contain mixed case data. The following example compares two Name fields and performs a function call when the names match:

```
If Upper(PERSONAL_DATA.NAME) = Upper(&EXTRACT_NAME) Then
   ProcessVerification(PERSONAL_DATA.NAME);
Else
   WinMessage("Names are not equal");
End-If;
```

Another commonly used string function is `Len`, which determines how many characters are contained in a string. To find out the number of characters stored in the field MY_PROBLEM_RESOLTN, the `Len` function can be used in the following manner:

```
&FIELD_LENGTH = Len(DESCRLONG);
```

17.2.10 Panel transfer functions

Functions in this category are associated with the transfer of control from one panel to another. PeopleCode programs using functions related to Transfer can be used to steer clients through related panels without prompting for key values. `AddKeyListItem` is a transfer function that can be used to help clients move around related panels without prompting for key values. Let's assume we are using the Operator Class/Locations application. The user links operator classes to locations and then links the operator classes to employee data. In order to set up a list of keys to facilitate transferring control to the next panel, `AddKeyListItem` can be implemented as follows:

```
AddKeyListItem(MY_LOCATION_EMP.SETID, MY_LOCATIONS.SETID);
AddKeyListItem(MY_LOCATION_EMP.OPRCLASS, MY_LOCATIONS.OPRCLASS);
AddKeyListItem(MY_LOCATION_EMP.LOCATION, MY_LOCATIONS.LOCATION);
```

A function used to transfer control when the operator presses F6 or presses the NextPanel toolbar icon is the `SetNextPanel` function. Continuing with the previous example, `SetNextPanel` can be used to transfer control to the panel MY_LOCATION_EMP after data on the operator class location panel is saved. The function verifies that the panel name appears on the current menu:

```
SetNextPanel("MY_LOCATION_EMP");
```

`SetNextPanel` identifies the panel to which control will be transferred following user actions such as F6 or activation of the NextPanel toolbar icon. The `TransferPanel` function is not activated based on user actions; it transfers control to the next panel in the panel group, the panel name supplied to the function, or the panel identified by a previous `SetNextPanel`. To illustrate the use of the `SetNextPanel` and `TransferPanel` functions, we make the assumption that the operator class location panels reside in the same panel group as the departmental security panels. The following statement transfers control to either the panel used to link locations and employees or the standard departmental security panel:

```
If %Panel = "MY_LOCATIONS" Then
   SetNextPanel("MY_LOCATIONS_EMP");
Else
   SetNextPanel("SCRTY_TABL_DEPT");
End-If;
TransferPanel();
```

An alternative method in which to code the `TransferPanel` function can be written using a variable set based on a conditional statement:

```
If %Panel = "MY_LOCATIONS" Then
   &NEXT_PANEL_NAME = "MY_LOCATIONS_EMP";
Else
   &NEXT_PANEL_NAME = "SCRTY_TABL_DEPT";
End-If;
TransferPanel(&NEXT_PANEL_NAME);
```

The example passes a variable to the `TransferPanel` function rather than using `SetNextPanel` to identify the next panel name.

17.2.11 Process Scheduler functions

PeopleCode can be used to submit a batch process that will run on a client or server location. A process can be associated with an SQR or a COBOL program and can be used to generate reports or processes, such as payroll batch cycles. `ScheduleProcess` is a function used to submit processes to the PeopleSoft Process Scheduler. The function accepts a number of parameters and stores a row of data into the process request table (PSPRCSRQST), which enables the system to schedule the process or job. The following example establishes the required and optional parameters to call the `ScheduleProcess` function. The example submits an SQR Report and identifies the process as `INSERTS`:

```
&PRCSTYPE = "SQR Report";
&PRCSNAME = "INSERTS";
&RUNLOCATION = "2";
```

```
&RUNCNTLID = "INSERTS";
&PRCSINSTANCE = "";
&RUNDTTM = %Datetime;
&RECURNAME = " ";
&SERVERNAMERUN = "PSUNX";
&RETURN_VALUE = ScheduleProcess(&PRCSTYPE, &PRCSNAME, &RUNLOCATION,
&RUNCNTLID, &PRCSINSTANCE, &RUNDTTM, &RECURNAME, &SERVERNAMERUN);
```

17.3 PEOPLECODE INTERNAL FUNCTIONS

An internal PeopleCode function is defined and used within the same PeopleCode program. The function can be declared in any PeopleCode program event, and the actual function definition requires that it be placed at the beginning of a program. Multiple functions can be defined within the same program, provided they are defined before any PeopleCode statements. In some functions, return values can be passed back to the calling program. Other types of functions perform some kind of operation on one or more fields.

17.3.1 Defining an internal function

An internal function is defined using the `Function` statement, which identifies the function name, parameters, and return value. A simple function definition can be written as follows:

```
Function MyFunction(&PARAMETER1, &PARAMETER2) Returns string;
End-Function;
```

The `Function` statement identifies the function name and any parameters passed to it. In the preceding example, the function `MyFunction` receives two parameters and returns a string. When a return value is specified, the data type of the returned value must also be specified. These return values can be any of the supported data types.

Example

An internal PeopleCode function can be defined and used in the Problem Tracking application. This internal function, illustrated by figure 17.4, will "live" in the `SaveEdit` event of the field MY_PROBLEM_STATUS. The purpose of the function is to verify that some text has been entered into the problem resolution field, when an issue has been resolved (&MY_PROBLEM_STATUS = "5"). The function also initializes the contents of the resolution text field (MY_PROBLEM_RESOLTN) when a value other than `5` is entered, and a Yes reply is entered from the message box.

The `Function` statement defines the function name as `MyTextFunction`. This function receives two parameters, `&PROBLEM_STATUS` and `&TEXT`. In the example, the fields passed are MY_PROBLEM_STATUS and MY_PROBLEM_RESOLTN. `MyTextFunction` returns a Boolean back to the calling routine as a return value.

The actual code contained in the function verifies if the variable `&PROBLEM_STATUS` has a value of 5 (resolved). If the value is resolved, the code then

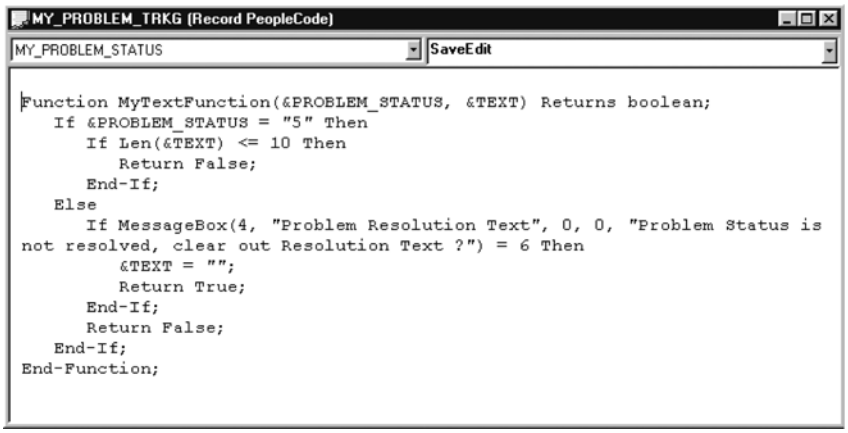


Figure 17.4 Internal PeopleCode function

checks the length of the variable `&TEXT`. Any length less than or equal to 10 produces an error message, which requires that a problem resolution text be entered. A length of 10 is used because there should be some type of dialogue that can be followed when reviewing problems and resolutions.

The other part of this function is enabled when a user had previously entered a problem resolution text and the problem is actually not resolved. With good intention, the user may have thought everything was working correctly, changed the problem status to 5, and entered comments into the resolution text field. Upon later review, however, this problem has not been resolved and requires additional analysis. When the value of MY_PROBLEM_STATUS is changed from resolved to another value, the user has the option of allowing the function to clear out the resolution text field or leaving it intact.

Figure 17.5 illustrates the use of `MyTextFunction` in the Problem Tracking application panel.

After the problem has been resolved, the problem status is changed to 5, and the problem resolution text is verified. The PeopleCode in the function statement verifies that a resolution text has been entered. An example of a resolved problem status is displayed in figure 17.6.

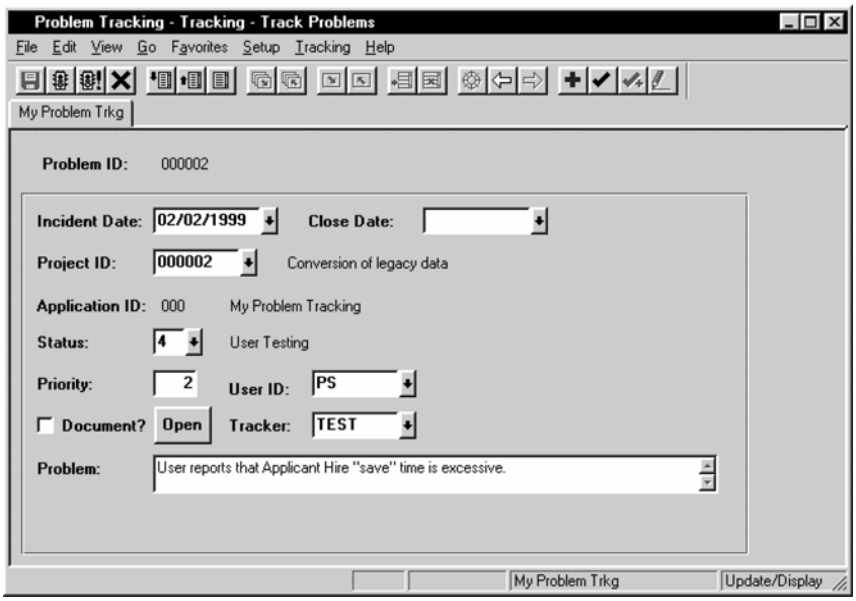


Figure 17.5 Example of an unresolved problem status

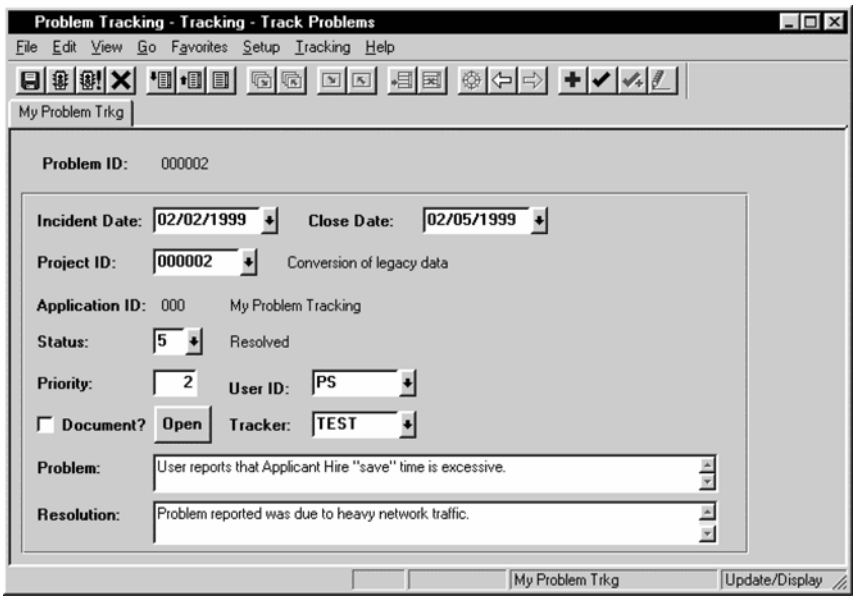


Figure 17.6 Resolved incident and resolution text

An additional feature of `MyTextFunction` provides the ability to clear out the problem resolution text when the status is prematurely set to 5 and subsequently set to another value at a later time. In the example, if the problem is not actually resolved and is changed to 4 (User Testing), a message (figure 17.7) is presented to the operator.

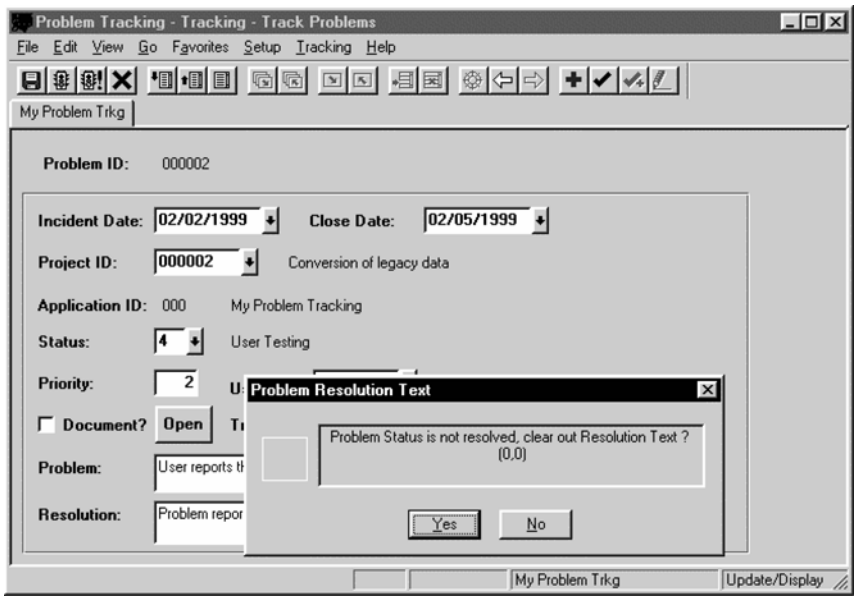


Figure 17.7 Internal PeopleCode function at work

Now that the problem status is no longer resolved, the existing problem resolution text may not necessarily apply after a final resolution. The PeopleCode in the function allows the user to keep the resolution text or remove it. The statement following the function call should reset the resolution text as well as the CLOSE_DT field. The statement can be written as follows:

```
If &RETURN_VALUE = True Then
   SetDefault(MY_PROBLEM_RESOLTN);
   SetDefault(CLOSE_DT);
End-If;
```

For PeopleCode internal and external functions, parameters are passed by reference. The `MyTextFunction` example in figure 17.4 contains the function call as follows:

```
&RETURN_VALUE = MyTextFunction (MY_PROBLEM_STATUS, MY_PROBLEM_RESOLTN);
```

The calling program passes the fields MY_PROBLEM_STATUS and MY_PROBLEM_RESOLTN. Because PeopleCode function parameters are passed by reference, any modification to the contents of either field contained in `MyTextFunction` are reflected in the calling program upon completion of the call. While the variables `&PROBLEM_STATUS` and `&TEXT` have unique names, they are actually pointers to the address of the fields MY_PROBLEM_STATUS and MY_PROBLEM_RESOLTN.

The internal function example presented in figure 17.6 is limited to the Problem Tracking application and, more specifically, to the MY_PROBLEM_STATUS.`SaveEdit` event for the problem resolution text field. A preferred objective is to write functions that can be used by more than one PeopleCode record field event. Nothing is lost when a function such as `MyTextFunction` is used, but there is also little to gain. Greater efficiencies can be realized by writing External PeopleCode functions, which are the next topic of discussion.

17.4 PEOPLECODE EXTERNAL FUNCTIONS

PeopleCode external functions do not have to be stored in the same record event as internal functions. The PeopleCode convention of storing functions places the code in Derived/Work records with either of two prefixes, `FUNCLIB_` or `DERIVED_`. Generally, these functions by convention are stored in the `FieldFormula` event. Two key items to understand when working with PeopleCode functions are defining a function and declaring one. A function such as `MyTextFunction` is defined in figure 17.4. Because the function is an internal PeopleCode function, it cannot be called from another PeopleCode program. However, if the function in figure 17.4 were an external PeopleCode function and called from another PeopleCode program, it would still be defined in the same manner. A PeopleCode function not defined in the calling program must be declared before it can be called.

Functions are useful when repetitive code is required and, more importantly, when the function can be called from various programs.

To effectively use external PeopleCode functions several steps must be followed:

1. Define the external function.
2. Declare the function.
3. Call the function.
4. Interpret any return values when necessary.

17.4.1 Define the External function

In figure 17.8 we have defined an external PeopleCode function. An externally defined function is defined in the same manner as a function called internally from the same Recordname.Fieldname.Event.

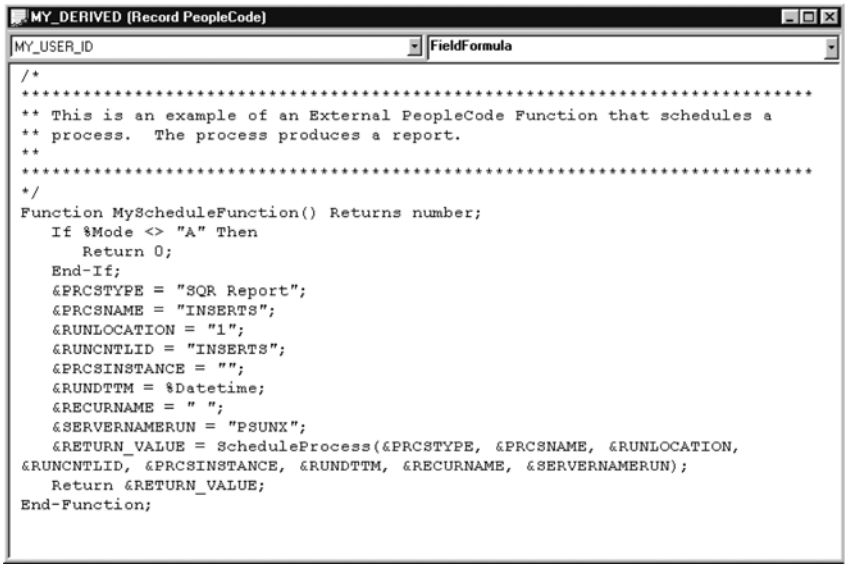


Figure 17.8 External PeopleCode function definition

A review of the code in figure 17.8 identifies the following:

- The function statement names the function `MyScheduleFunction` and indicates that the function receives no parameters but returns a number value.
- The statements contained within the function verify that the current mode is `Add` before continuing.
- A function can call another function. In the example, `MyScheduleFunction` is calling the built-in function `ScheduleProcess` and passes parameter values initialized by the statements preceding the call.
- The function returns a zero when the mode is not `Add`. The `ScheduleProcess` function returns a value that is also the return value passed back by `MyScheduleFunction`.
- `End-Function` signals the end of the PeopleCode function. This statement is required for both internal and external function definitions.

17.4.2 Declare the function

Now that we have a function defined, it would be great to call it occasionally. Before a PeopleCode function can be called, however, it must be declared in the calling program. A `Declare Function` statement is required for each unique function that it calls. If a program calls five distinct functions it will require five function declarations. The function declaration identifies the function name—and where it resides—in terms of record name, field name, and event. A `Declare Function` statement can be written as follows:

```
Declare Function MyFunction PeopleCode MY_DERIVED.MY_USER_ID FieldFormula;
```

A PeopleCode function can be defined on any record definition. All PeopleCode function declaration statements must appear at the top of the calling program before any of the regular program code.

Example

For the Problem Tracking application, we would like to execute a process each time a new user is added to the database.

The external function is defined in figure 17.8. With the exception of comments, which can appear anywhere in a program, the `Declare Function` statement(s) must be at the top of the calling program. When adding new users to MY_USER_TBL, the function call can be inserted into the `SavePostChg` event and written as shown figure 17.9.

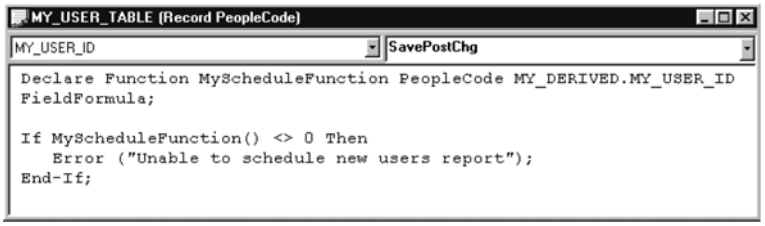


Figure 17.9 Declaring and calling the external function

The code illustrated in figure 17.9 uses the `Declare Function` statement to identify the function name as `MyScheduleFunction`. The additional parameters specify that it is a PeopleCode function and resides in the record field MY_DERIVED.MY_USER_ID. The function is stored in the `FieldFormula` event. The function call statement is used in the context of a conditional `If`, which interprets the function call return value. A non-zero value issued by the function call implies an unsuccessful call, which generates an error message.

17.4.3 Call the function

In the example presented in figure 17.9, the `MyScheduleFunction` call does not pass any parameters. An external PeopleCode function call can simply be written as `Function_Name()`. Any parameters passed are enclosed in parentheses.

17.4.4 Interpret return values

Many circumstances require that the return value of a function be interpreted before proceeding with the remaining code. A return value can be any data type. A common convention is to return zero when the function call is successful or neutral. The example used in figure 17.9 utilizes the return value in the context of an `If` statement. The external function `MyScheduleFunction` returns a zero when the mode is not `Add`. This type of verification could certainly be done before the function is called, so that no function call is made from modes such as `Update/Display` or `Correction`. When writing functions, however, it is important to be prepared for all types of parameters and circumstances. While one developer may call `MyScheduleFunction` during `Add` mode only, other developers may call it, regardless of mode. Similarly, some functions may require data types such as `Number` or `String` exclusively. Such functions should be written in a manner that can handle incorrect data and pass the appropriate return codes that can be useful when the need for debugging arises.

The `Return` statement is used to transfer control from the current active function back to the calling program. After the calling program has received control, program execution resumes with the next logical statement following the function call.

17.5 EXTERNAL NON-PEOPLECODE FUNCTIONS

Another type of function to consider is an external Non-PeopleCode function. This type function is declared differently from a PeopleCode function and is stored in a C-callable library. PeopleCode can call an external program, which may be useful to us when a required complex function already exists in a C-library or when we wish to interface with Windows (DLL) or equivalent UNIX accessible Dynamic Link Libraries (shared libraries/shared objects). The idea here is to take advantage of these libraries when the function already exists outside of PeopleCode or when the need for performing unusual or behind-the-scene types of tasks becomes necessary. Some of these functions can interact with the operating system, system hardware, or perform operations that require programs at levels such as C++, JAVA, or Assembler. When calling external Non-PeopleCode functions, the `Declare Function` statement is somewhat different than the external PeopleCode declaration.

Example

An External non-PeopleCode function declaration can be written as:

```
Declare Function OpenTextFile Library "My_Lib.dll" Alias "OpenFile"
      (string, integer) Returns integer;
```

The Declare Function statement identifies the function name as OpenTextFile. The function resides in a Dynamic Link Library named My_Lib.dll. The optional Alias name is OpenFile.

In the example, the ext_datatype parameters which refer to the data types the function expects are identified as String and Integer. Any parameters passed to the external function are enclosed within one set of parentheses. The Return statement indicates the function returns an integer.

The function declaration example can be written in a more complex manner by identifying optional parameters:

```
Declare Function OpenTextFile Library "My_Lib.dll" Alias "OpenFile"
      (string Ref As string, integer Value As number) Returns integer As
number;
```

The preceding example contains additional parameters. One parameter in particular may impact program results. The parameter list identifies the first parameter expected by the called function to be a String. Additionally, REF indicates it is passed by reference, which implies that the address of the data element is passed to the called function. The pc_type of the first parameter identifies it as being a String data type in the calling PeopleCode program. The called function expects the second parameter to be an integer. Its pc_type is a number in the calling program. The potential impact is that the second parameter is passed by Value. As previously discussed, passing a parameter by Value signals that the actual value is passed to the called function. How can this impact a program? When a value is passed by reference (REF), the address of the data element is passed to the called function. Any subsequent modifications made to that data element in the called function are reflected when control is returned to the calling program. Specifying Value passes the actual value of the data element; however, any changes made to the data element in the called function are not reflected in the calling program after control is returned to it. Consequently, a calling program which expects modified data following a function call, will not receive modified data when the parameters are passed by Value.

A non-PeopleCode external function call is identical to the other types of PeopleCode function calls. For the code presented in the preceding examples, the first parameter represents a file name and its corresponding path, passed as a string. The second parameter identifies the manner in which the file will be opened. The '1' represents Open for Input and a '2' represents Open for Output. The function call is shown as:

```
&FILENAME = "MyInput.txt";
&IO_TYPE = 1;
&RETURN_VALUE = OpenTextFile(&FILENAME, &IO_TYPE);
Rem  Verify Return Value 0 = Ok;
If &RETURN_VALUE <> 0 Then
   Error ("File " | &FILENAME | " Cannot be opened");
End-If;
```

KEY POINTS

1. A function is a collection of programs or subroutines which, when called, perform a specific task.
2. A PeopleCode function can be stored on any record event.
3. Internal PeopleCode functions are defined within the same record event in which they are called.
4. Callable PeopleCode functions are referred to as external PeopleCode functions and are defined in a record event. The function call passes any variables to the called function.
5. PeopleCode external functions can be shared by different programs and can improve efficiency by reducing program code.
6. All functions are defined with a `Function` statement, regardless of whether they are internal or external.
7. The `Declare Function` statement is required when calling a PeopleCode or non-PeopleCode external function.
8. External non-PeopleCode functions are not commonly used, but can help to perform tasks that cannot be easily done by standard PeopleCode functions.

CHAPTER 18

PeopleCode debugging tools

If you are new to PeopleTools and have read, understood, and applied the techniques and information provided thus far, then you are on your way to becoming a PeopleTools developer. Of course, your introduction will not be complete and should not be complete until you have encountered problems with records, panels, and PeopleCode. A good developer is one who writes code that works, is efficient, and can be reused. A great developer is one who can use debugging tools and knows how and where to look for bugs.

18.1 THE FIRST BUG

According to computer industry folklore, the first stored program computer was invented in 1944 by the U.S. Army; it was called EDVAC. As fate would have it, the first reported computer "bug" was a moth, which was caught up in the computer and discovered by a U.S. naval officer and mathematician, Grace Murray Hopper. Webster's *New World Dictionary of Computer Terms* defines a bug as "a mistake in a computer program or system, or a malfunction in a computer hardware component. To DEBUG means to remove mistakes and correct malfunctions."

As with nature, computer bugs come in all sizes. Minor computer bugs cause little inconvenience; more serious bugs can impact a payroll or financial posting process; and real serious bugs can create catastrophes similar to the NASA Mars Climate Orbiter lost in 1999. That problem was initially reported as either human error or software error. Most likely, there is no difference. At this point in time, computer programs which can "think" for themselves are quite basic and mostly left to Hollywood films. To some extent, computer bugs will always exist. Even HAL 9000 was (or will be?) bug-ridden. Our objective as developers is to minimize bugs, and when they do arise, limit their impact and know how to go about resolving them. Knowing how to detect bugs and use debugging tools are most important.

18.2 USING WINMESSAGE

`WinMessage` can be a useful debugging tool because it can be used to display information in a message box window. A `WinMessage` statement containing an OK button can be used in any PeopleCode event. If more than one button is used, the function becomes a `user think-time` function limited to specific PeopleCode events only. With the exception of an `Object` data type, `WinMessage` converts any data type into a string and displays it in the message box window.

NOTE Refer to chapter 14 and appendix E for additional information regarding `WinMessage`.

Debugging a program using `WinMessage` enables PeopleCode to display the contents of variables, system variables, and record fields. A constant can be displayed alone or concatenated with multiple data elements.

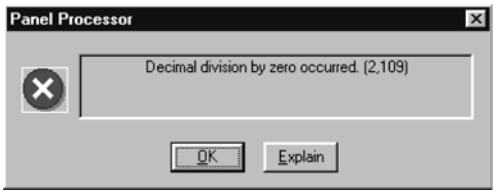


Figure 18.1 Error message returned

Let's assume we have a panel group comprised of six panels. Each panel requires multiple data elements and fires off many calculations. When an error message similar to the one in figure 18.1 is issued, `WinMessage` can be used to help locate the code and data responsible.

The suspect code can be identified by inserting WinMessage statements into the PeopleCode programs linked to the events in question. This can be accomplished by inserting a simple message to identify the program and event.

To identify the suspect code, WinMessage statements can be written as follows:

```
WinMessage("This is JOB.COMPRATE.SaveEdit");
WinMessage("Now in DERIVED_HR.COMPRATE.FieldFormula");
```

After the program is identified, WinMessage can then be used to narrow down the set of statements generating the error. Let's assume the following statements are suspect:

```
If &A > &OLD_RATE Then
   &NEW_RATE = ((&X + &Y) / (&Z * &OLD_RATE));
End-If;
```

WinMessage can be used to combine string constants and variables into one statement.

For example:

```
WinMessage("The value of &X is " | &X | " Y = " | &Y | "  Z= " | &Z);
```

From a debugging perspective, WinMessage is a simple debugging tool used to locate and identify PeopleCode programs and data elements. In some applications of WinMessage, however, the process of identifying and localizing bugs may appear redundant and sometimes tedious. Complex functionality which involves many PeopleCode programs, functions, and data elements may be more difficult to track using WinMessage. A more intuitive debugging PeopleTool which is flexible and takes some of the guesswork out of tracking down errors, is the Application Reviewer, which is the next topic of discussion.

18.3 THE APPLICATION REVIEWER

The Application Reviewer is used to trace the path of PeopleCode programs as they execute. It enables the developer to stop a program at a specific point in its execution, list the contents of variables, and trace external or internal function calls. Program return values and PeopleCode function parameters can also be logged. Features of Application Reviewer include establishing breakpoints, viewing data elements, and tracing programs, data, and program calls.

After acquiring a good working knowledge of Application Reviewer, the task of debugging can be simplified.

18.3.1 Breakpoints

A breakpoint is a preset area of code where the program will stop execution. When setting breakpoints with Application Reviewer, we have the opportunity to inspect code and view the contents of variables or record fields. Stepping through

PeopleCode and identifying what effect each statement has on variables and fields can be easily accomplished using Application Reviewer. Alternatively, we can review all the PeopleCode associated with a particular panel group. This can be done using the Break at Start menu option.

The Application Reviewer works on one panel group at a time. As a result, the Application Reviewer is opened after the panel group that needs to be debugged has been started.

The modification of a user name in Problem Tracking can be used to track PeopleCode programs using Application Reviewer. First, start the menu of the application to be debugged. In this example, the Problem Tracking menu is started (figure 18.2).

Navigation: Go →Problem Tracking

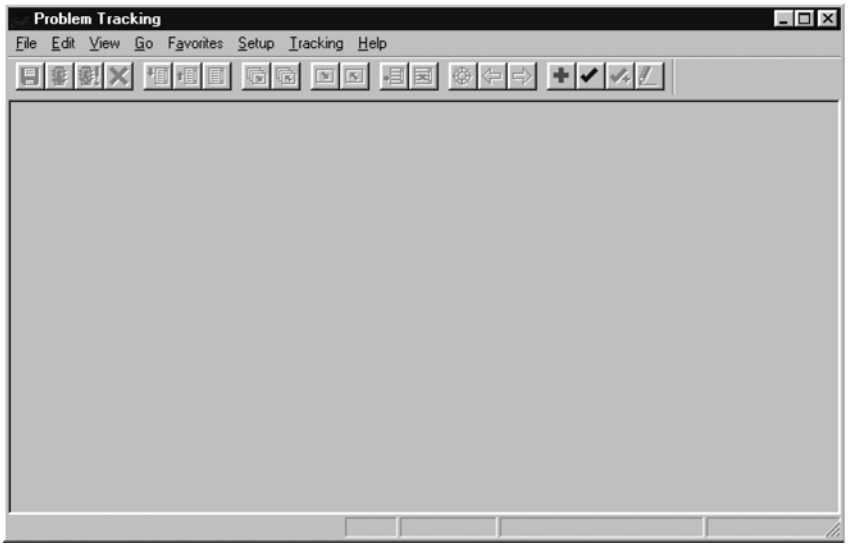


Figure 18.2 Start panel before application reviewer

The display of the Problem Tracking menu is then followed by the display of the Application Reviewer menu (figure 18.3).

NOTE The Application Reviewer panel is not available in the PeopleTools menu until after an application has been started

---

Navigation: Go →PeopleTools →Application Reviewer

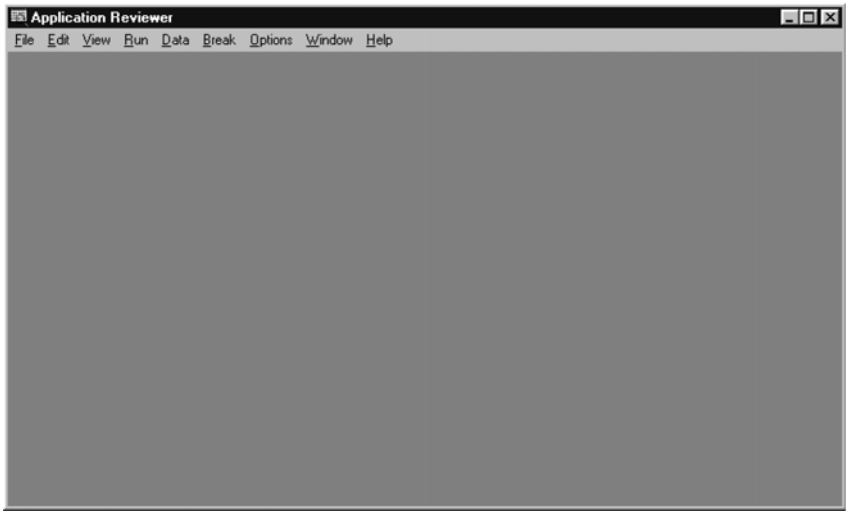


Figure 18.3 Initial Application Reviewer panel

Break at Start

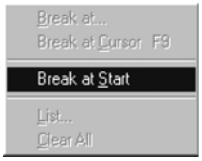


Figure 18.4
Break at Start

To establish a generic breakpoint after the Application Reviewer menu is displayed, choose Break, Break at Start as shown in figure 18.4.

The next step is to enter the specific application panel group that is being debugged. When Break at Start is selected from the Application Reviewer menu, any PeopleCode executed at panel startup is displayed. This includes PeopleCode events such as `SearchSave` and `SearchInit`.

TIP For a better understanding of PeopleCode and event processing, refer to chapter 13.

The steps required to use break are now set. Be aware, however, that if the break is set correctly and no PeopleCode break point is detected, it is possible the PeopleCode programs may only execute after the panel is displayed. Events executed after the panel group is displayed include `FieldEdit` and `FieldChange`. Consequently, the Application Reviewer is displayed when the first PeopleCode program is detected and not necessarily when the application panel is displayed.

Navigation: Problem Tracking →Setup →Users

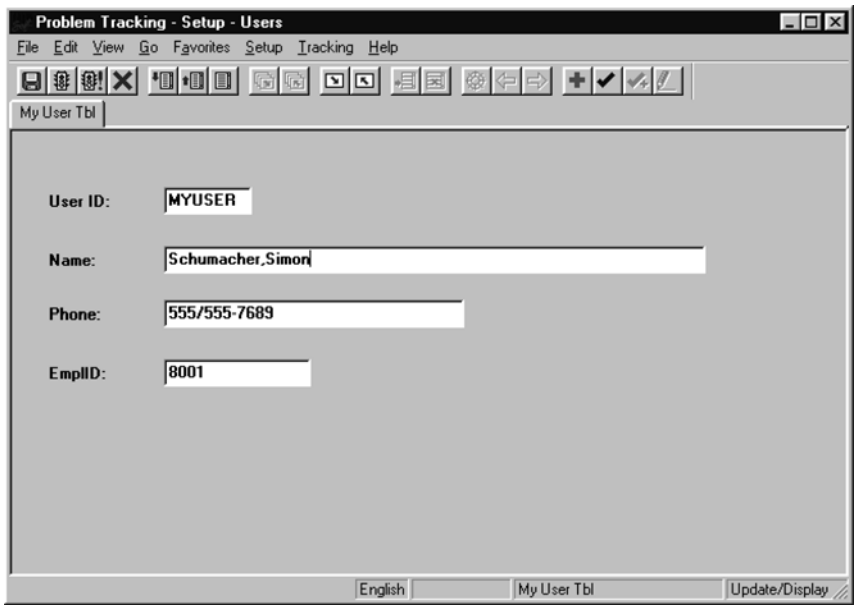


Figure 18.5 Setup Users Panel

For the example presented, no breaks have been encountered so far. We can see by the illustration in figure 18.6 that PeopleCode exists for this record in `FieldEdit`, `SaveEdit`, and `SavePostChg`.

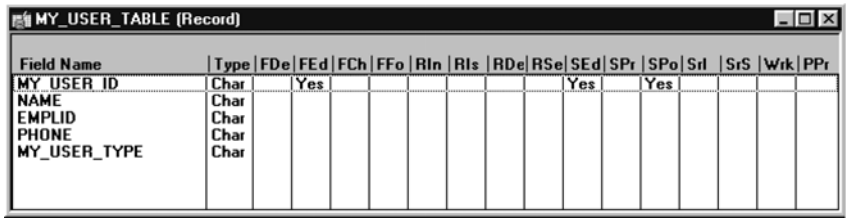


Figure 18.6 PeopleCode events for MY_USER_TABLE

The PeopleCode illustrated in figure 18.6 explains why no breaks have been displayed up to this point. It is worth mentioning that, when PeopleCode exists in an event such as `FieldDefault`—where we expect to have a break before the panel is displayed—the PeopleCode may not necessarily get displayed. Recall that any `FieldDefault` PeopleCode is not executed for record fields containing data when

a record is subsequently displayed. `FieldDefault` PeopleCode is, however, an iterative process that continues to execute until a field has a value.

We have changed the name field for user `MYUSER`. After tabbing out of the field or pressing the save button, the Application Reviewer screen is displayed as shown in figure 18.7.

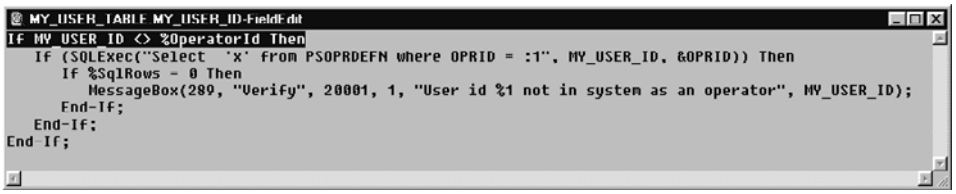


Figure 18.7 Application Reviewer window

Several options are available after the Application Reviewer panel is displayed. Pressing F4 or using the View, Show Panel menu item enables a jump between the user panel and Application Reviewer.

Figure 18.8 Application Reviewer Run menu items

The Run menu option, shown in figure 18.8 contains several Step features which add flexibility when stepping through PeopleCode.

When Break at Start is used, F5 or Run →Go enables the current program to execute until the next PeopleCode program is detected. After breakpoints have been specified throughout the program, pressing F5 resumes processing until the next break in the program is found.

The F7 or Run → Step to Cursor steps through the PeopleCode until the cursor location.

The F8 or Run →Step is used to execute the current line of PeopleCode and step into functions. In the example in figure 18.7, the PeopleCode verifies if MY_USER_ID, entered from the User ID panel (figure 18.5), exists as a valid ID on the Operator security record. If the ID does not exist, as detected by the `SQLExec` statement, the PeopleCode sends out a message using `MessageBox`. When stepping through code, the message is displayed, and no more stepping is allowed until the message box is closed. In the example, we have an OK and Cancel button based on the style parameter. Pressing OK allows the Application Reviewer to continue.

The F10 or Run →Step Over option steps through the line of code but does not step into functions. The functions are still executed, but with this option they are not stepped into.

The Run →Step Return menu option stops after the current function has returned.

Run →Step Instr is used with the Log view and allows simulated machine code instructions to be processed.

Break at Start is one method of setting breakpoints. Establishing one or multiple breakpoints in one or more PeopleCode programs can be accomplished using the Application Reviewer.

Break at Cursor

Using Break at Start on a panel with many fields and corresponding PeopleCode programs may require stepping through a range of programs and events. A specific breakpoint can be set in one or more programs using Break at Cursor. This can be accomplished as follows:

Navigation: Application Reviewer →File →Open →MY_USER_TABLE

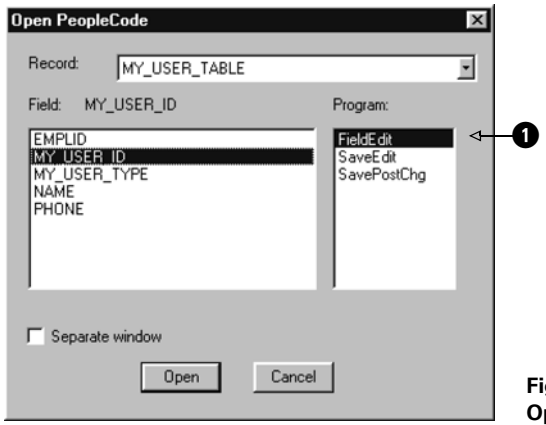


Figure 18.9
Opening a record field event PeopleCode

The PeopleCode events associated with MY_USER_ID are shown in figure 18.9, ❶. For this example, the `MY_USER_TABLE.MY_USER_ID.FieldEdit` event is selected. The PeopleCode for the event is presented in figure 18.10.

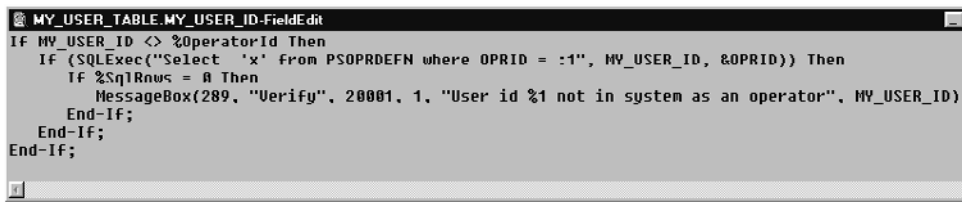


Figure 18.10 PeopleCode opened for Break at Cursor

To establish a breakpoint, move the cursor to the line of PeopleCode on which to break. Breakpoints can then be set using one of the following methods:

- double-clicking on the cursor line,
- pressing F9, or
- selecting Break at Cursor from the Application Reviewer menu.

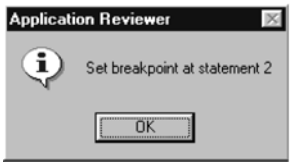


Figure 18.11
Breakpoint confirmation

In the example, the breakpoint is set on the second line of the program. The message in figure 18.11 is issued, which indicates the breakpoint is set on the second statement. When a breakpoint is set, the same options available when using Break at Start apply to Break at Cursor as well. These include stepping through the PeopleCode after a break is encountered.

Navigation: Application Reviewer →Break →List

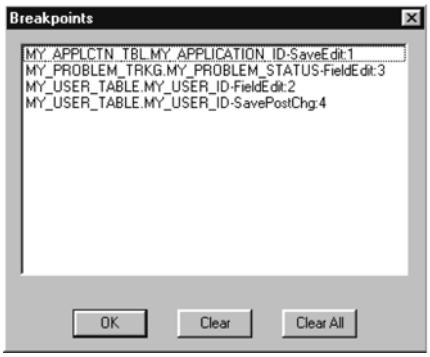


Figure 18.12 Breakpoint dialog box

Additional breakpoints set in the current PeopleCode program or other events in the Problem Tracking panel group can be identified using the Break, List option. List is used to display the breakpoint dialog box, which contains the breakpoints for records, fields, and events contained in the panel group.

Breakpoints can be cleared in two ways. One manner is to clear all breakpoints using the menu Break, Clear All, which removes all breakpoints. To clear one or more particular breakpoints and keep the remaining ones intact, use the Breakpoints dialog box.

18.3.2 Viewing data

Navigation: Application Reviewer →Data →PeopleCode Variable

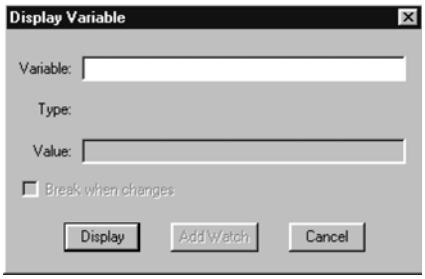


Figure 18.13 PeopleCode variable dialog box

Viewing data elements can also be accomplished using Application Reviewer. This is very useful when debugging because it enables us to see the impact that PeopleCode statements and functions have on data. Some functions have a black box effect, but with Application Reviewer, the return values of functions can be examined. After a break is executed, data can be viewed using the Application Reviewer menu shown in figure 18.13.

The previous section illustrated how to set breakpoints in a PeopleCode program. A breakpoint was set on statement 2 (figure 18.11), which is the `SQLExec`.

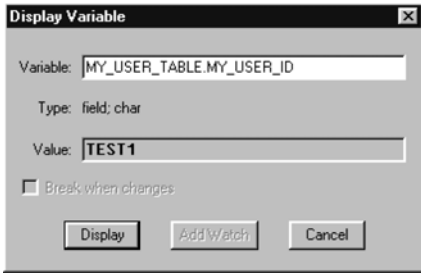


Figure 18.14 Value of MY_USER_ID field

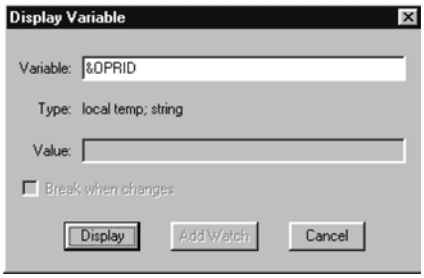


Figure 18.15 Value of &OPRID variable

To view the value of the record field MY_USER_ID, the display variable dialog box can be entered as shown in figure 18.14.

Characteristics of the Display Variable dialog box include the record fieldname, data type, and value of the field. Upon conclusion of the `SQLExec` statement, the contents of variable `&OPRID` can also be displayed. What is the value of the variable `&OPRID` after `SQLExec`?

The `&OPRID` variable (figure 18.15) does not contain a value because `SQLExec` selects the literal `x` and, therefore, the variable is not populated even when the `Select` statement finds a match. Using a literal in a `Select` statement is a common practice when all that is required is the verification of data without the need for return values. This is pointed out because it is important to understand what the PeopleCode statements and function calls are doing. In this particular example, knowledge of SQL is important to avoid inaccurate conclusions when it is discovered that `&OPRID` does not contain a value.

18.3.3 Additional Application Reviewer options

Local or global variables can be viewed using the View Locals menu option, which displays the Local Temps screen illustrated in figure 18.16.

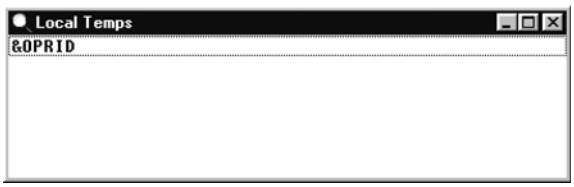


Figure 18.16 Viewing local variables

Application Reviewer can be used to view log files which reflect selected system activity. This can be set by selection of the menu option as shown in figure 18.17.

Navigation: Application Reviewer →Options →Log

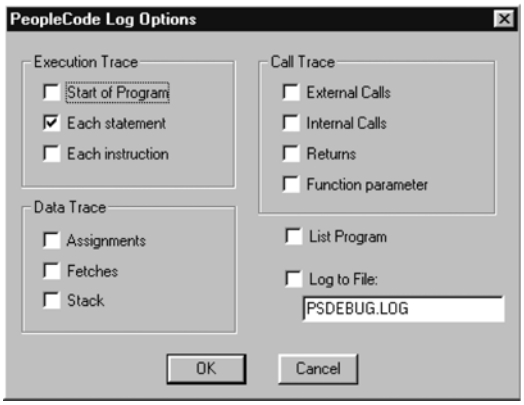


Figure 18.17
PeopleCode log options

As illustrated in figure 18.17, PeopleCode log options are divided into three categories:

- Execution Trace
- Data Trace
- Call Trace

The Execution Trace option enables PeopleCode programs to be traced at the start of each program, each program statement, or each program instruction. The default is set to each statement.

The next option available is Data Trace. The options are Assignments, Fetches, and Stack. Assignments log all data assignments made to variables or record fields. The log window in Application Reviewer must be open to log data assignment activity to a file. Some caution must be used when logging activity. It is possible to generate very large log files during a PeopleSoft session. Logged traced data assignments are written to the default file PSDEBUG.log. A typical log window may look like the following:

```
7
  0 If MY_USER_ID <> %OperatorId Then
>> Begin MY_USER_TABLE.MY_USER_ID-FieldEdit
^^^^^^^^^^ PeopleCode Program Listing End
 396, stop
 377, statement Next=396
 376, pop
 364, builtin    - MessageBox #Parms=6
 348, fetch     MY_USER_TABLE.MY_USER_ID
 298, push      User id %1 not in system as an
 274, push      1
 251, push      20001
 234, push      Verify
 212, push      289
```

```
 202, branch <>  377
 180, push      0
 164, fetch     Builtin - %SqlRows
 152, br  False  377
 140, builtin    - SQLExec #Parms=3
 126, push      &OPRID (temp #0)
 112, push      MY_USER_TABLE.MY_USER_ID
  60, push      Select  'x' from PSOPRDEFN whe
  52, branch =   377
  36, fetch     Builtin - %OperatorId
  20, fetch     MY_USER_TABLE.MY_USER_ID
   1, statement Next=377
   0, start     Field=MY_USER_TABLE.MY_USER_ID-FieldEdit Temps=1 Stack=6
vvvvvvvvvv PeopleCode Program Listing:
```

The panel activity is displayed in the log window. To write the contents to a file, a File Save operation is required before closing the Application Reviewer. Depending on the log options set, a significant amount of information is stored in a trace log file. Some of this information requires further interpretation.

A log file can be viewed using an ASCII text editor such as PFE. The components in a log file include

- line number in the file
- internal tracing reference numbers
- address of instructions in a program
- operation code
- operation operands represent information used by each operation

The operation code and operation operands work in conjunction with one another and are available to the list and trace options. The operation code refers to the internal operation performed by the program. The operand is the value required by the specific operation. Some operation codes and operands used in the preceding code example are as illustrated in table 18.1:

Table 18.1 Operation codes and operands

Operation	Operands	Description
Start		identifies the beginning of a PeopleCode program
Push	constant	pushes the operand into the stack
BR True	location	This instruction works with the internal program stack. It checks the current item on the stack for a Boolean return value. When the value is `True`, program control is transferred to the operand location.
Branch	location	Program control is transferred to the operand location. No return value is tested.
Fetch	Record.field	retrieves a record field value and pushes it onto the stack
BR False	location	works with the internal program stack (It checks the current item on the stack for a Boolean return value. When the value is `False`, program control is transferred to the operand location.)

Table 18.1 Operation codes and operands (continued)

Operation	Operands	Description
BuiltIn	Function #Parameters	executes the specified function represents the number of parameters
Call	DLL	calls a routine stored in a Dynamic Link Library
	Internal function	calls an internal PeopleCode function
	External function	calls an external PeopleCode function stored in record.fieldname
Error		halts the PeopleCode program
Exit		exits the PeopleCode program
Return		returns control to the next higher level program
Store	record.field	stores the top stack item into the operand record field
Stop		identifies the end of a PeopleCode program
Pop		Removes top item from the stack

We can use the information in table 18.1 to examine selected log file entries found in the preceding code example.

Line #	Details of operation performed
348	The Fetch Operand retrieves the record.field operand and pushes it onto the stack. In this example, the record.field is `MY_USER_TABLE.MY_USER_ID`
298	The `Push` statement is used to move the constant to the stack.
202	The branch `<>` statement compares the two top items on the stack. In this example, control is passed to line 377. This specifies the location of the next statement, which is a `Stop` operation.

18.4 SEARCH IN PEOPLECODE

Application Designer contains a search facility that can be used to look for strings contained in PeopleCode programs. Search can be performed against record PeopleCode, menu PeopleCode, or both. The search can be tailored to look in a specific record or several records. The search can also be used to search all PeopleCode programs. When searching for an exact string a Match Case option exists. The results are sent to a search output window, with the additional option of sending the output to a file. To begin searching, select Edit →Find in PeopleCode from the menu.

The dialog box (figure 18.18) is used when searching the entire database for the `MessageBox` string. Searching the entire database is necessary when no additional information is known about the location of the PeopleCode string. Clicking on the Define List button can be used to narrow a search. This produces the Define Search List window (figure 18.19) where the number of records or menus to search can be limited to specific records or prefixes. Using this feature produces more efficient results.

Navigation: Application Designer →Edit →Find in PeopleCode

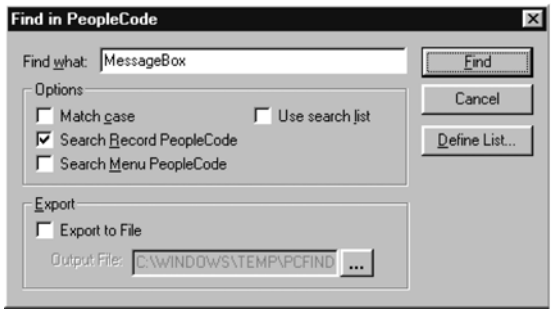


Figure 18.18
Find in PeopleCode dialog box

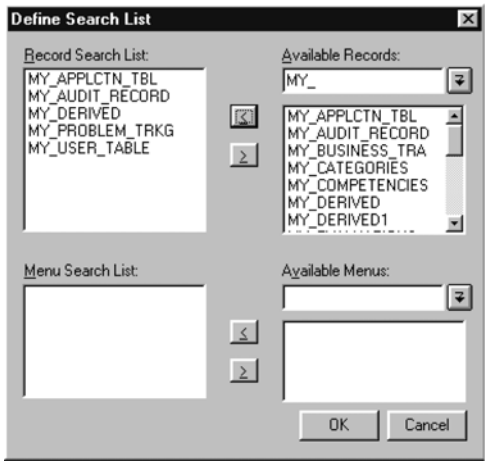


Figure 18.19 Define Search List dialog

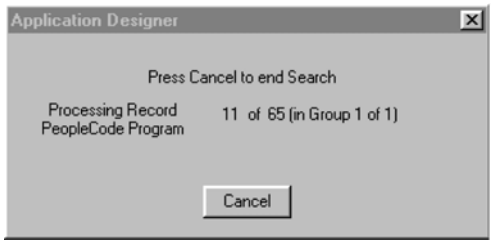


Figure 18.20 Search status

The search can be narrowed to a specific record key—in this example, record names that begin with MY_. A list of available records containing MY_ as the record name prefix is used. The selected records are moved to the left side of the dialog box. Clicking OK closes the Define Search List dialog. The Find button is then pressed, and the search begins. The status is displayed in figure 18.20.

When a search list is not specified, the search may run for several minutes, depending on the number of records searched. A search which specifies a list containing many records may also require additional time. Pressing the Cancel button while performing a search can stop the search process.

The search results are displayed in the Find in PeopleCode output window. The results of the search can also be sent to a file for later viewing. This can be done by clicking the "Export to File" checkbox in the Find in PeopleCode dialog box.

18.5 PEOPLECODE TRACE

PeopleCode Trace is another tool used for debugging. Trace produces an output file defined in the trace page of Configuration Manager. Two methods of producing a PeopleCode Trace file exist: the Trace PeopleCode menu option and the `SetTracePC` Function.

18.5.1 Trace PeopleCode utility

The Trace PeopleCode feature enables the tracing of PeopleCode programs using preset options in the Trace PeopleCode panel.

The illustration in figure 18.21 identifies how several trace options can be set before beginning a Problem Tracking panel session. The trace file is a standard text file that can be opened and read with a file editor such as PFE or Notepad. The trace file contains the operation codes and operands generated by the program. The file is similar to the log file discussed in the Application Reviewer section. There is good reason for this similarity: the trace utility can be replaced by the Application Reviewer trace without losing trace option functionality.

A second trace method is the `SetTracePC` function. This function helps control PeopleCode Trace settings from one or more PeopleCode programs.

Navigation: Utilities →Use →Trace PeopleCode

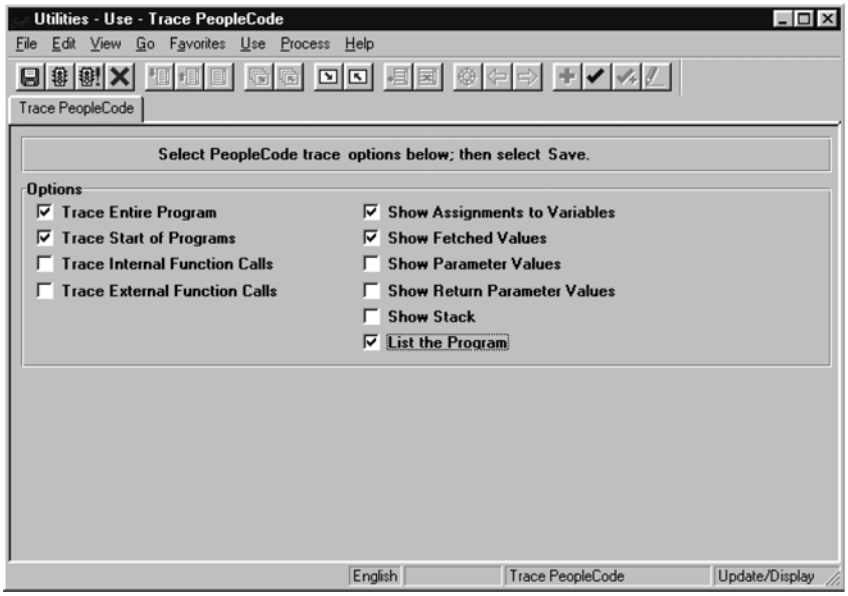


Figure 18.20 Trace PeopleCode utility

18.5.2 SetTracePC

The SetTracePC function controls PeopleCode Trace, based on parameter values passed to the function. The function takes one parameter, which represents the trace settings used in producing the output trace file. If multiple trace options are required, each option number is added, and the sum is passed to the SetTracePC function.

SetTracePC produces a file named DBG1.tmp in the Windows Temp directory. We can specify a unique name if necessary and this can be done from within the configuration manager trace option. The Set Trace options are as follows:

Table 18.2 SetTrace PC options

Value	Option description
1	This option traces the program that is executed. It includes options 64, 128 and 256 specified below.
2	Lists the entire program.
4	Displays the outcomes of assignments made to variables.
8	Identifies the values retrieved for all variables.
16	Identifies the contents used in the internal stack.
64	This trace option identifies when each program is started.
128	Identifies the calls made to external PeopleCode routines.
256	Identifies the calls made to internal PeopleCode routines.
512	Displays the value of parameters passed to a function.
1024	This option displays the values of parameters at the conclusion of a function call.

Based on the values identified in table 18.2, a statement using SetTracePC can be written as follows:

```
&OPTION_VALUE = 14;
SetTracePC(&OPTION_VALUE);
```

In the example, the option value passed to SetTracePC generates a file containing the following:

- Program listing +2
- Results of variable assignments +4
- Values of variables fetched +8

At this point, we have a better understanding of how to use PeopleCode program debugging tools.

KEY POINTS

1. `WinMessage` is a function which can be used to help in debugging by displaying the contents of variables and record fields. `WinMessage` can be used in any PeopleCode event provided only one button exists in the message box.
2. The Application Reviewer is a key tool used in debugging PeopleCode programs. With Application Reviewer we can set breakpoints and examine the results of PeopleCode statements and function calls. The value of record fields and variables can also be displayed.
3. The Application Reviewer can be used to identify local and global variables.
4. Trace options in Application Reviewer can be applied to program execution, data, and function calls.
5. Search PeopleCode is an Application Designer tool that enables the search of strings contained in PeopleCode programs. The search can be done across the entire database or against specific records.
6. PeopleCode Trace is a utility that can also trace PeopleCode programs. It produces a report and can be initiated from the Utilities menu option or from within a PeopleCode program using the `SetTracePC` option.

CHAPTER 19

PeopleCode—PeopleSoft 8

With release 8 of PeopleTools and, more specifically, PeopleCode, we have vast amounts of new knowledge to acquire. In regard to PeopleCode, features in release 7.5 are not lost. Backward compatibility and the integration of new PeopleCode functionality enables PeopleCode to work with features available in release 8. Release 8 contains key enhancements to the PeopleCode language and the environment in which these new tools are used.

In this chapter, we briefly discuss some new items which make the development and implementation of PeopleCode more exciting and challenging than ever before. As the World Wide Web becomes a major player in business and personal use, PeopleCode is there with Web Client and Internet Client design.

An attempt to list and describe the new features in PeopleCode release 8 would practically require a small book on its own. Some of the topics selected for this

discussion of release 8 were reviewed in previous chapters. This should provide the reader some perspective between releases 7 and 8. PeopleCode release 8 is also very object oriented. As a result, the File and SQL objects are briefly illustrated. Additional topics include new panel events, enhanced scroll processing and working with arrays. In release 8, a tool that we will use at one time or another is the PeopleCode Debugger. The debugger includes increased functionality and can be entered directly from the Application Designer.

19.1 FILE OBJECT

One of the more important features in PeopleTools 8 is PeopleCode object syntax. Standard classes and the use of dot notation enable us to access functions and objects contained in these classes. The use of object syntax allows PeopleCode to execute in Application Engine programs not linked to panel groups. This signals a move away from the relationship between PeopleCode and panel groups. PeopleCode can now be written so that it is run in a stand-alone mode. This C++ type syntax also contains classes comprised of methods (which are the functions contained in a class). One such class is the File class, which contains methods (functions) and properties (fields) that can be used to open, read, and write to external or "flat" files. If you are a C++ or Visual Basic developer, you'll appreciate these classes.

To define a File object, the statement can be coded as:

```
Local File &MyTextFile;
```

This example creates a file object named `&MyTextFile`, which is a pointer to an object that contains file-handling methods. Because the variable is a pointer, the file object is passed by reference to PeopleCode functions and methods. Now, let's assign the File object an address of a file contained on a floppy disk:

```
&MyTextFile = GetFile (a:\"Interface.txt", "R");
```

The statement calls the `GetFile` function used to create a new instance of a file object based on the File class. The function links the file object with an external file and then opens the file Interface.txt. After the file is opened, additional methods contained in the File class can be used to read from or write to the file. In the example, the second parameter passed to the `GetFile` function represents the mode in which the file is read. An "`R`" indicates the file is opened for Reading.

We now wish to read the file into a string using the `ReadLine` method. `ReadLine` is a File object-associated method that reads a line of text from the file object. In the following example, text is read into the string `&Interface_Record` and passed to an external PeopleCode function named `ProcessInterfaceData`:

```
Local String &Interface_Record;
While &MyTextFile.ReadLine(&Interface_Record);
```

```
   &RETURN_VALUE = ProcessInterfaceData(&Interface_Record);
End-While;
```

To close and unlink the file object `&MyTextFile` from the external file Interface.txt, we can use the `Close` method, which frees up all resources connected with the file object:

```
&MyTextFile.Close();
```

19.2 SQL OBJECT

Earlier releases of PeopleCode used the `SQLExec` statement to perform database operations such as `Select` or `Update`. For release 8 of PeopleTools, SQL definitions can be created in the Application Designer and can subsequently be used as SQL programs. Components of the SQL statements can subsequently be re-used. In PeopleCode 8, programs can now access SQL definitions through the SQL class. While the `SQLExec` function can still be used in release 8, the SQL class has added functionality which enables multiple rows to be selected. When performing a `Select` using `SQLExec`, the function only selects the first row. The SQL class is similar to `SQLExec` because it supports bind values and output variables.

The following example defines an SQL object named `&MySQL`. The `CreateSQL` statement is used to create an instance of an SQL object, which is opened based on the values passed in the SQL string.

`CreateRecord` is a method associated with a Record object. Data rows will be selected from the record in this example:

```
/*  Define SQL and Record objects */
Local SQL &MySQL;
Local Record &MyRecord;

&MyRecord = CreateRecord(Record.MY_LOCATIONS);
&MySQL = CreateSQL ("%Selectall (:1) where SETID = :2 and OPRCLASS = :3");
```

Our next step is to execute the SQL statement of the object `&MySQL`. This statement `Selects` from the record MY_LOCATIONS and loads the corresponding fields into `&MyRecord`, which is an object of MY_LOCATIONS:

```
/* Execute the attached statement */
&MySQL.Execute (&MyRecord, &SETID, &OPRCLASS);
```

The next statement is `Fetch`, which is a method associated with an SQL object. The `Select` statement in the preceding example retrieves the rows, based on the value of bind variables supplied during execution of the statement. The following `Fetch` operation retrieves each subsequent row processed by the Select statement associated with `&MySQL` objects.

```
/* Fetch from the select row */

If &MySQL.Fetch(&MyRecord) Then
 ProcessMyLocationRecord(&MyRecord);
 End-if;
```

After data rows have been processed, it is necessary to close the SQL statement. The `Close` method associated with an SQL object disconnects `&MySQL` from the `Select` statement. The `Close` statement is illustrated as follows:

```
/* Close the SQL object */
&MySQL.Close();
```

19.3 ASSOCIATING PEOPLECODE WITH PANEL GROUPS

Some PeopleCode used throughout this book—more specifically, the programs contained in fields shared between two or more panels—may contain statements which identify the panel name. These statements are executed before any processing is performed. The code used by a particular panel must first be prefixed with an `If` statement:

```
If %Panel = "MY_LOCATIONS" Then
/* Statements associated with this panel */
End-if;
```

The preceding code is associated with specific functionality for the panel only. A similar `If` statement, related to another panel with a different set of statements, can also reside in the same record field event. This approach works but is usually not efficient, particularly in terms of record reusability. In release 8, PeopleCode can be linked to panel groups and the elements that make up a panel group such as panels, panel records, and panel fields. The PeopleCode associated with panel groups and their corresponding components include:

- Panel group record field PeopleCode
- Panel group record PeopleCode
- Panel group PeopleCode
- Panel PeopleCode
- Panel field control PeopleCode

This type of PeopleCode is only available from a panel group definition and the events for the specific panel group components. Panel group record field PeopleCode is different from record field PeopleCode because the former is associated with record fields which exist on a panel group and its related events. Panel group PeopleCode is not available from a record definition; it can only be retrieved through a panel group's structure. An example of a panel group structure and corresponding panel group record field PeopleCode is illustrated in figure 19.1.

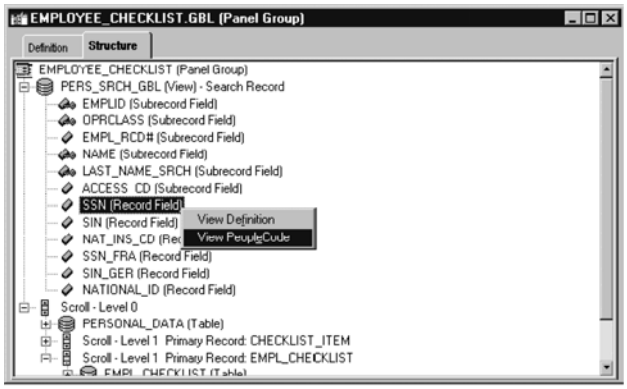


Figure 19.1
Panel group record field PeopleCode from the panel group structure

Panel groups contain several new events associated with panels and panel groups. The events are `Activate`, `PreBuild`, and `PostBuild`.

19.3.1 Activate event

When a panel is initially displayed the `Activate` event is triggered. The event is also generated when tabbing between panels contained in a panel group. PeopleCode which resides in this event is specific only to the panel and can include some of the functionality related to panel display, such as `Hide` and `UnHide`. Each panel contains its own `Activate` event. PeopleCode in this event can only be linked to panels.

19.3.2 PreBuild

`PreBuild` PeopleCode is related to panel groups only and is triggered before the remaining panel group build events are executed. PeopleCode in this event can be used to hide or unhide panels.

19.3.3 PostBuild

PeopleCode in this event is triggered after the other panel group build events have been generated. Programs in this event are linked to panel groups only and can be used to set panel group variables.

19.4 ENHANCED SCROLL FUNCTIONS

In chapter 16, we learned how to implement PeopleCode scroll functions such as `ScrollSelect` and `RowScrollSelect`. In release 8, the new `Rowset` class can be used to work with `ScrollSelect` or `RowScrollSelect` using the class methods, `Select` and `SelectNew`. These class methods enable the PeopleCode program to control the selection of data into panel scroll areas. `Select` and `SelectNew` are used with a `Rowset`, which is equivalent to a scroll area. The level zero area of a panel is also the level zero `Rowset`. The level zero `Rowset` also contains data in the panel buffers. In chapter 16, we learned about parent and child relationships.

Release 8 uses child Rowsets controlled by a higher level Rowset known as the parent Rowset. The process of selecting into Rowsets also includes child Rowsets when autoselect is enabled. Child Rowsets, however, are processed in a manner similar to RowScrollSelect and utilize a WHERE to limit child Rowsets to that of the parent Rowset.

19.4.1 Using Select

The Select method associated with a Rowset class is used to retrieve rows from an SQL table or view. The record definition of the SQL table or view from which rows are retrieved is referred to as the select record. The top level Rowset containing the PeopleCode which executes the Select is referred to as the default scroll record. Select automatically positions child Rowsets under the corresponding parent row executing the method. As with ScrollSelect and RowScrollSelect, Select also accepts an optional SQL string which can include a WHERE block used to limit the rows into the scroll area.

Let's apply the Rowset class and its corresponding Select method to the level 2 ScrollSelect example presented in chapter 16.

The first step creates an instance of a record object named &MY_LOCATIONS_REC and a Rowset object, based on the MY_LOCATIONS record. The PeopleCode is shown as follows:

```
Local Record  &MY_LOCATIONS_REC;
Local Rowset &MY_LOCATIONS;
```

The next step (illustrated following) uses GetRecord to create a record object which references the MY_LOCATIONS record. The GetRowSet method is used to create a Rowset object and the current MY_LOCATIONS row.

```
&MY_LOCATIONS_REC = GetRecord(RECORD.MY_LOCATIONS);
&MY_LOCATIONS = GetRowSet (SCROLL.MY_LOCATIONS);
```

The Select method is implemented using the child Rowset MY_LOCATION_EMP, which is passed as the first parameter to the Select method. The next parameter is the select record represented by the view MY_LOC_EMPL_VW. The new Meta-SQL function %KeyEqual extends into a conditional phrase that can be used in the WHERE clause. When more than one key exists in the record, the phrase will include an AND for each of the record keys. %KeyEqual performs the task of automatically applying the Meta-SQL functions %Datein(), %TimeIn() and %DateTimeIn(), based on the data type of the field. When the value is a string it will be enclosed in quotes. A NULL value will be replaced with "IS NULL". The bind variable (:1) passed to %KeyEqual is the record object &MY_LOCATIONS_REC.

```
&MY_LOCATIONS.Select (SCROLL.MY_LOCATION_EMP, RECORD.MY_LOC_EMPL_VW, "WHERE
%KeyEqual(:1)", &MY_LOCATIONS_REC);
```

NOTE `SelectNew` behaves like `Select` except that `SelectNew` marks records as new.

19.5 ARRAY CLASS

The PeopleCode Array class enables the definition of array objects without necessarily specifying a fixed size. The array size expands and contracts based on whether data are added to or removed from the object. A simple array can be declared as:

```
Local Array of String &MyStringArray;
```

Multi-dimensional arrays can also be specified. A two-dimensional array comprised of numbers can be declared as:

```
Local Array of Array of Number &MyNumberArray;
```

Referencing elements in an array can be accomplished using indexes. The array `&MyStringArray` can be referenced as follows:

```
&String_Element = &MyStringArray[5];
```

19.5.1 Populating an array

An array can be populated several ways. On method uses `CreateArrary` when the array is initially created and can be written as:

```
&MyStringArray = CreateArray("String 1","String 2","String 3");
```

An array can also be populated by assigning values to the individual array elements:

```
&MyStringArray = CreateArray();
&MyStringArray [1] = "String 1";
&MyStringArray [2] = "String 2";
&MyStringArray [3] = "String 3";
```

`Push` is a method associated with an array class:

```
&MyStringArray.Push("String 1");
&MyStringArray.Push("String 2");
&MyStringArray.Push("String 3");
```

`Unshift` is another method associated with an array class. It can be used to add items to the beginning of the array:

```
&MyStringArray.Unshift ("String 1");
&MyStringArray.Unshift ("String 2");
&MyStringArray.Unshift ("String 3");
```

19.5.2 Removing items from an array

Two methods associated with an array class include `Pop` and `Shift`. To select and remove array items from the end of an array using `Pop`, the code can be written in the following manner:

```
&String_Field = &MyStringArray.Pop();
```

In the preceding example, the value of `&String_Field` is `"String 3"` after using the `Pop` method.

The `Shift` method can be used to select and remove items from the beginning of an array:

```
&String_Field = &MyStringArray.Shift();
```

Because `Shift` targets the beginning of an array, the value of `&String_Field` is `"String 1"` following execution of the `Shift` method.

19.5.3 Using an array in a loop

A basic PeopleCode `For` loop can be written to reference the values in an array. A numeric data element can be utilized as an index to reference array items. The code can be written as follows:

```
For &I = 1 to  &MyStringArray.Len;
   &String_Field = &MyStringArray [&I];
End-For;
```

`Len` is an array class property that represents the current number of items in an array.

19.6 PEOPLECODE DEBUGGER

The PeopleCode Debugger, formerly known as Application Reviewer, offers increased functionally and simplicity of use. In release 7, the Application Reviewer is entered using Go →Application Reviewer. Now, the PeopleCode Debugger is included in the Application Designer. A panel group can now be started, and the PeopleCode Debugger can be entered from Application Designer. Alternatively, another panel group can be entered and automatically run in debug mode.

19.6.1 Improved visual support

In chapter 18, we saw how Application Reviewer enables breakpoints to be set so that PeopleCode statements can be tracked. The PeopleCode Debugger includes a visual indicator of breakpoints and an arrow that identifies the current line of code. Figure 19.2 shows an arrow which illustrates how the current line of code is identified, ❶.

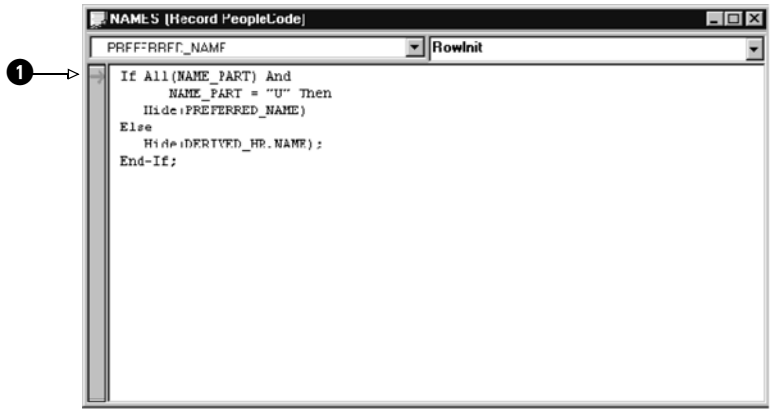


Figure 19.2 Identification of current line executing

The PeopleCode Debugger also visually identifies breakpoints using indicators as shown by ❶ in figure 19.3.

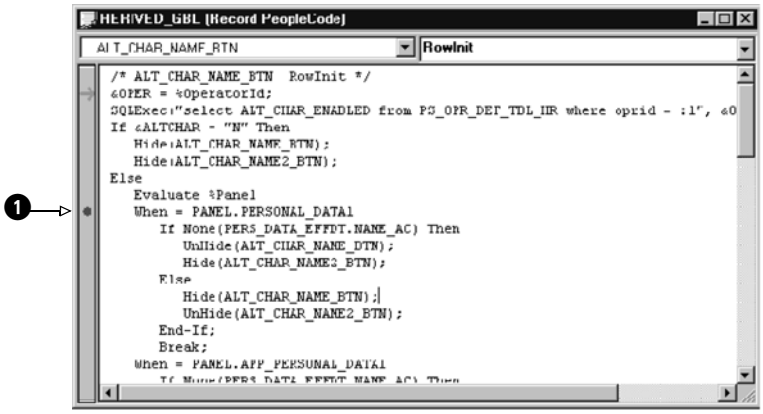


Figure 19.3 Visual indicator of breakpoint

Another great feature of the PeopleCode Debugger is referred to as "Hover Inspect." This feature enables the visual inspection of simple variable or field contents displayed in a pop-up window. An example of using Hover Inspect is illustrated in figure 19.4. The contents of the variable after the data assignment is denoted by ❶.

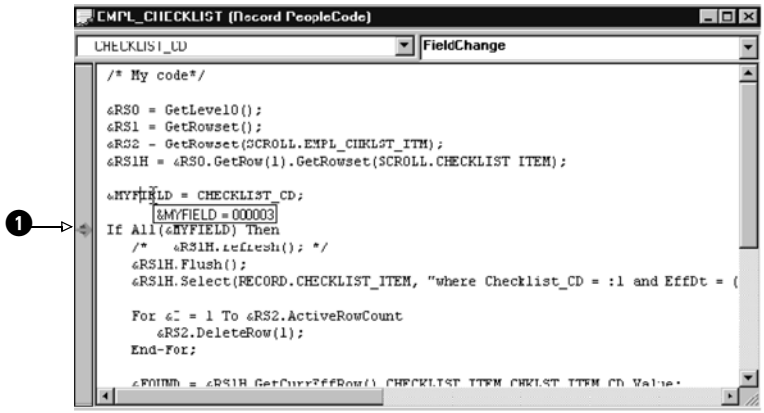


Figure 19.4 Using Hover Inspect

Variables can now be viewed from different windows based on the type. These variable types and their associated window include Local, Global, Panel Group, and Parameter. The Parameter window can be used to view user-specified parameters included in function calls. An example of a Local variable window is shown in figure 19.5.

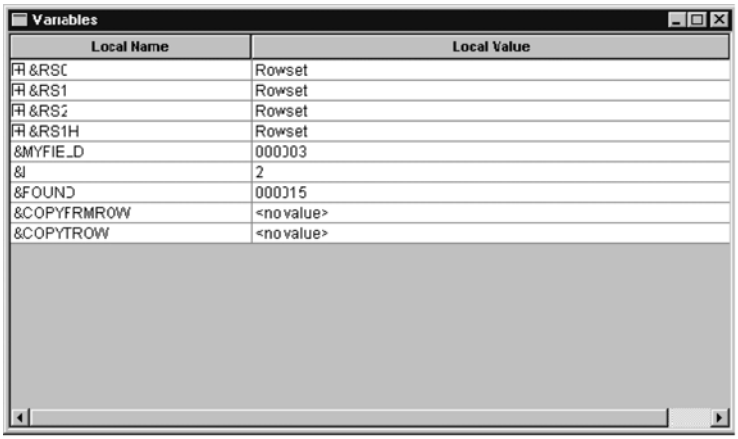


Figure 19.5 Local variable window

The visual representation of objects can be extended to display the object properties. The "+" convention is used and appears next to the variable name. The example in figure19.5 identifies `Rowsets` at several levels. The level 1 `Rowset` is expanded

(figure 19.6) to reveal properties that comprise the `Rowset`. Some properties include Effdt, Name, and `ActiveRowCount`.

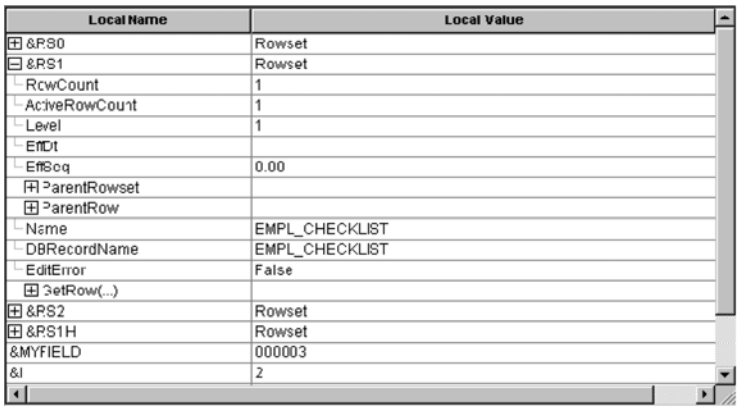


Figure 19.6 Expanded Rowset object

Field object values are listed under the value column when viewed in the debugger. The field value can now be viewed without having to navigate to the value properties. Figure 19.7 illustrates how the PERSONAL_DATA record and its associated fields can be viewed.

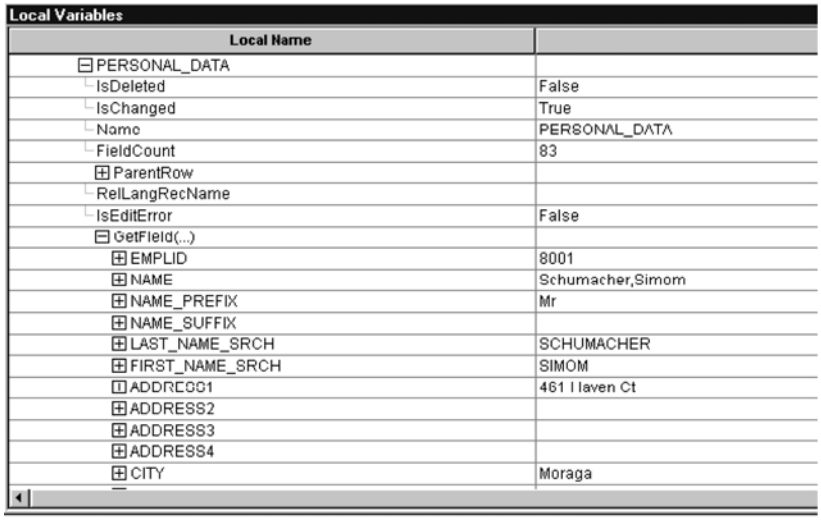


Figure 19.7 Viewing record field values

19.6.2 Additional options

Additional options which can be selected after the debugger is running include:

- Exit Debug Mode
- Abort Running Program
- Edit Breakpoints

The Exit Debug Mode option automatically saves all breakpoints before leaving debug mode. The PeopleCode program currently running can be terminated using the Abort Running Program option. The Edit Breakpoints option displays a menu identifying the lines containing breakpoints (figure 19.8).

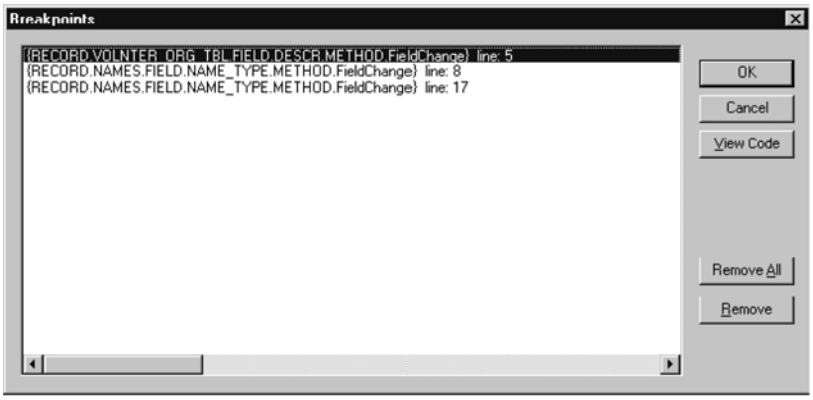


Figure 19.8 Edit breakpoints menu

This menu enables us to remove specific breakpoints or to remove all breakpoints. The View Code button will display the code containing the breakpoint. The menu also identifies the line number in the PeopleCode program which contains the breakpoint.

PART 4

Customizing PeopleSoft-delivered applications

To be recognized as one of the leading contenders in the ERP arena is no small feat. Each packaged application produced by the ERP vendor must have functionality to accommodate a wide range of companies with varying business requirements. Some requirements may be dictated by the particular laws of a given state or even country. With the expansion of companies into the global marketplace, the challenge to provide complete ERP solutions is greater than ever. PeopleSoft has responded to these challenges and has produced an impressive set of functional applications in areas such as Finance, Human Resource Management, Manufacturing, and Student Administration. These applications may provide customers with all the functionality they need right-out-of-box. This is known as running plain "vanilla" PeopleSoft. Some companies, especially the larger variety, are inundated with rules, regulations, and requirements unique to their businesses. These companies are faced with a decision: to either re-engineer their business processes or customize the PeopleSoft applications. The philosophies and perceptions toward customization vary. Some companies avoid customization at all costs while others embrace the opportunity. In many cases, you will find no other alternative exists but to customize the delivered applications.

PeopleSoft provides a full set of tools to create and manage these customizations. We discuss these tools in detail while providing information on the customization process, project management, and software upgrades. The reader can follow along with sample customizations while maintaining a project that contains all of the modified objects. Through the use of PeopleTools,

customers are also afforded the opportunity to create application extensions and subsystems which provide increased functionality while having relatively little impact on upgrades. It is always in your best interest to fully understand the impact and management of customizations to get the most out of PeopleSoft.

CHAPTER 20

"Vanilla" vs. customized

20.1 WHAT IS CUSTOMIZATION?

As you've seen in the previous chapters, PeopleSoft delivers a great suite of tools that are simple to use and that allow for speedy development cycles. Some of our readers are probably anxious to start the development and customization using PeopleTools. But is this always necessary? And what kind of considerations should we always keep in mind?

All of us in the PeopleSoft world have probably heard the frequently used term "Vanilla." Strictly speaking, this means no customizations are allowed to the delivered system. In a more liberal interpretation, this means you should customize only when necessary. The perpetual dilemma—to customize or not to customize—remains. Both approaches present advantages and disadvantages (which we will discuss) but as always in real life, a reasonable compromise, in most cases, is the best way to go. Our goal after all is to make the new system functional, convenient, and user-friendly. First let's

agree on what is considered a customization, what kind of customizations may be necessary, and what kind of customizations are better to avoid.

A customization is any change to a delivered application. Let's divide the entire pool of all customizations into additions to the PeopleSoft-delivered application (or system extensions); and true modifications to these applications.

Imagine a situation when your old system had a functionality not present in the PeopleSoft-delivered system. You have the following choices to consider:

- change your company business practice and drop the old functionality
- develop a manual desk procedure to support the old functionality
- create an external (Excel, Access, Visual Basic, and so forth) application that can interface with the delivered PeopleSoft application
- change PeopleSoft-delivered objects and programs to incorporate the desired functionality into the delivered application
- develop an addition to the delivered application by creating your own menus, panels, records, and processes utilizing PeopleTools.

Let's first briefly characterize each option. Later, we will discuss some of these alternatives in greater detail.

20.1.1 Changing your company business practice

Often (this is more common to small and midsize companies), management tries to save on both system development and future maintenance by changing the current business rules and processes in order to fit the PeopleSoft-delivered system. Generally speaking, implementing PeopleSoft creates a good opportunity to review current processes and procedures and bring them up to industry standards. PeopleSoft has done considerable research and has built systems that are supposed to address all possible needs of an average company. In reality, not all businesses are able to fully adapt the delivered applications. The problem is that this "average" company is an abstraction. It simply does not exist. Real companies have traditions. They may follow unique business practices that separate them from other businesses in the industry and (who knows?) may help them to compete on both labor and primary business markets. Implementing PeopleSoft-delivered systems already means big changes and a tremendous psychological impact, but it should not also involve cutting off important and healthy business functionality just because aspects don't fit the "Vanilla" option.

20.1.2 Developing a manual desk procedure

Can we keep the good old functions and still spare ourselves the development cost? Let's make every function that is not available in the delivered system a manual desk procedure! This may be the perfect approach for seldom-used tasks, but if the function has to be performed on a regular basis, automation is the way to go.

20.1.3 Creating a satellite application with interface to PeopleSoft

Oftentimes, the management realizes that changes are inevitable, but may still insist on keeping the system "Vanilla." They may then try to augment the PeopleSoft-delivered modules with satellite applications using available desktop computing tools such as Microsoft Excel, Access, and so on. Developing satellite applications just to avoid any changes to the PeopleSoft system may not always be the best choice. Instead of maintaining one centralized PeopleSoft system, you may end up maintaining a number of different applications and interfaces between them. On the other hand, if a satellite system already exists and has all the functionality that users need, developing an interface to PeopleSoft may be a good idea.

20.1.4 Changing PeopleSoft-delivered objects and programs

In some cases, changes to the PeopleSoft-delivered objects are absolutely necessary to suit business needs. This option comes with a hidden price tag. Later in this chapter, we will discuss in greater detail the considerations you would have to bear in mind to minimize the impact on a delivered application. The most important consideration is a future upgrade. What happens when PeopleSoft delivers a new release? Your changes then have to be merged with the new application release. Some changes may be straightforward, and others may be more complex. A simple task such as adding a new field to an existing PeopleSoft table, for example, may be a dangerous exercise. In the subsequent chapters we'll discuss what to look for as well as how to minimize impact on the delivered application.

20.1.5 Developing additions with PeopleTools

If added functionality can be independent of the PeopleSoft-delivered system, our preference would be choice number five: Developing a subsystem by using PeopleTools. With PeopleTools, you can develop entire subsystems to supplement the "Vanilla" application while keeping them separate from the delivered application. Why is it so important to keep new objects isolated from the delivered ones? Well, one reason is to minimize the impact on future application upgrades. Another reason is to prevent changes made to the application from interfering with the delivered PeopleSoft application.

20.2 UPGRADE CONSIDERATIONS

What is an upgrade to a new PeopleSoft release, and why should this be an important consideration when making changes to the delivered application?

We all know that software applications constantly go through upgrades, improvements, and even total restructuring and redesign to be competitive, but how do these changes affect customizations? Usually, a good software package makes its releases backward-compatible. The problem is that PeopleSoft not only delivers software development tools, it also delivers an entire suite of applications. In addition, PeopleSoft

usually delivers a new major PeopleTools and application release once a year. Sometimes, a PeopleTools release is delivered twice a year in order to compete with other software companies. PeopleSoft has a strict schedule that you would have to follow in order to keep your software package current and be supported by PeopleSoft.

It is important to understand the concept of upgrade procedure in order to make good and educated decisions on customizations.

PeopleSoft did its best to develop good tools that are designed to perform upgrades to new releases. Nevertheless, the process of upgrades remains cumbersome. The most difficult part of the upgrade process is usually comparing the currently used application to the base application of the same release and to the new release of the application. Even though the compare programs will identify the differences for you, you still have to go over every change and decide—and this is an extremely important decision—whether to carry a change over to the new release, drop the change, or merge the change with the PeopleSoft changes. This last option is the most difficult. It happens when the changes to the object are done by both you and PeopleSoft.

All in all, the fewer changes you make to the delivered application, the less time required to perform an upgrade.

Although a detailed discussion of all steps involved in a release upgrade is not in the scope of this book, we want to describe ways to make the maintenance of PeopleSoft applications easier.

Rule 1. When performing system modifications, document every change. For example, if you have changed an existing Record definition or created a new one, put your initials, Date/Time stamp, type of modifications (`New`, `Change`), and a brief explanation in the comments section (figure 20.1).

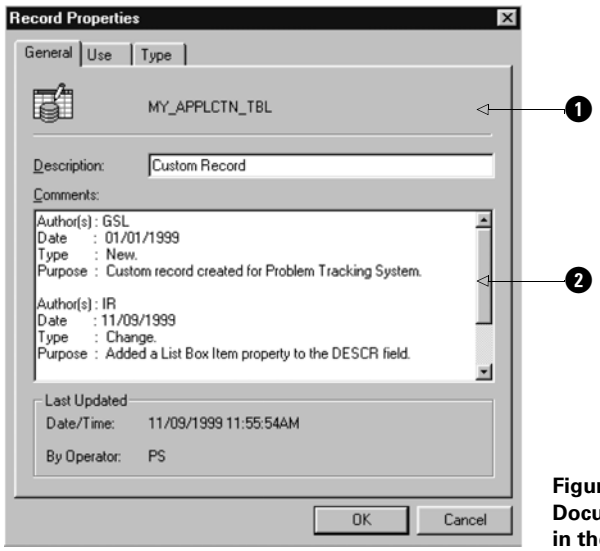


Figure 20.1
Documenting a change in the record properties

❶ The record has a prefix of MY_ to identify a customization.

❷ The header specifies the initials, the Date/Time of customization, the modification type, and the short description.

Always insert extensive comments when changing PeopleCode programs. It is a good idea to develop a standard change header and use it all the time when performing customizations. It should include the developer's name, date/time, reason for change, and any other useful information. Developers must also put in trailers to mark the end of any changes they have made when customizing PeopleCode or any other program (figure 20.2).

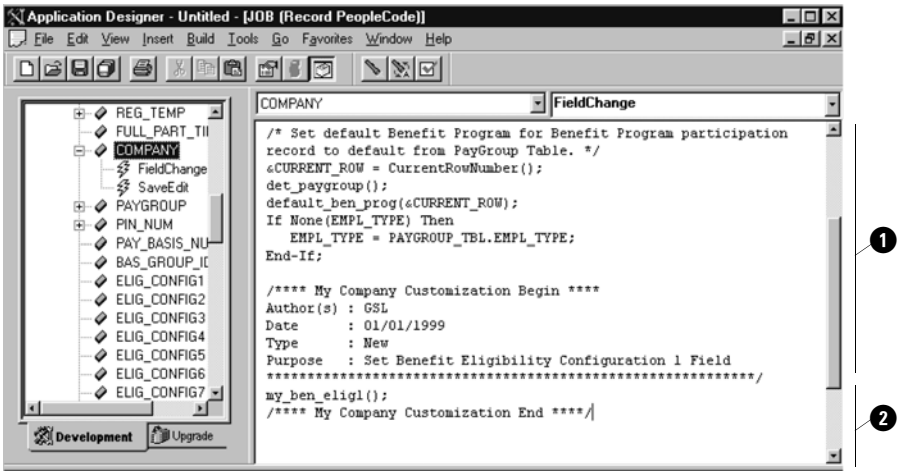


Figure 20.2 Documenting PeopleCode changes

❶ The header marking the beginning of PeopleCode customizations.

❷ The trailer marking the end of PeopleCode customizations.

When you prefix changed objects with certain characters, such as your company name or abbreviation, you can easily identify all the changes you or your colleagues have ever made. You can use any prefix letters that suit your needs.

In our Problem Tracking application developed in part 2, we used the prefix MY_ to identify all custom records, fields, panels, and so on. A new record definition may include both the existing and custom fields. Hence, if you prefix all custom fields, you can identify the changes immediately.

Rule 2. Use a prefix to identify your custom objects. For example, in figure 20.3 you can see that we named our custom record as MY_APPLCTN_TBL and the custom field as MY_APPLICATION_ID. The other two fields in this record are PeopleSoft-delivered fields (figure 20.3).

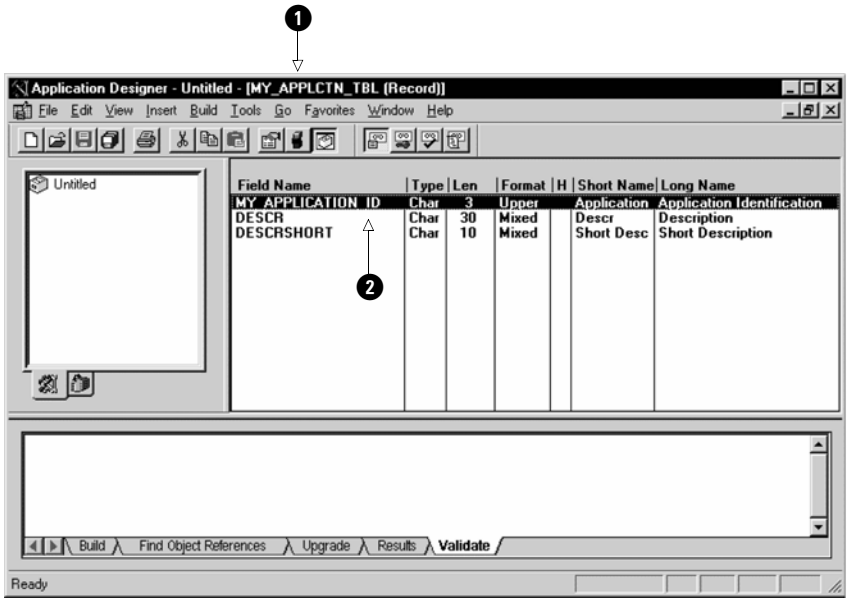


Figure 20.3 Using a prefix to identify custom objects

1. Prefix "MY" is used to identify custom record
2. Prefix "MY" is used to identify custom field.

Carefully examine the change requirements and always consider different ways to accomplish the customizations. If there is a way to isolate the changes, consider making them an addition to the system rather than changing the delivered system.

Let's suppose a user requests that a new field be added to an existing panel. As a PeopleSoft developer, you know that the related record may have to be modified as well. Be careful. A simple change like this can sometimes lead to major problems once a new version of PeopleSoft is released. Consider a situation when you just need to add one custom field to an existing record definition. During an upgrade to a new PeopleSoft release, you would have to add this field to the Record definition and alter the table. But this is not the only thing you would have to consider. What if the new release includes an SQR or COBOL program that uses the same table and inserts a record into this table? Since all fields in PeopleSoft tables (except date fields) are defined as `NOT NULL` fields, the SQR or COBOL program delivered by PeopleSoft will result in the following database error: `'Inserting a NULL value to a column where NOT NULL specified'`. Here's why: When the above-mentioned program inserts a record into the modified table, it does not know anything about your modifications. As a result, the program only inserts the values into the specified table columns, causing the database engine to insert `NULL` values into the custom columns. To avoid this

problem, you have to search all programs for any possible inserts into the changed table and modify the programs as necessary.

Another way to perform the same changes is to create a new custom record and add a custom panel to an existing panel group. In this case, the initial work requires more effort, but later during the upgrade you only need to be concerned with adding another panel to the panel group. Is this the recipe for all customizations? Not necessarily, because every particular case may have its own twists and each should be considered individually. If, for example, you have simple panel changes to apply, it may make more sense to perform modifications in place. In this case, you can take advantage of the Upgrade Compare process that identifies the changes you performed. (We will discuss all of these and more in the subsequent chapters.)

Rule 3. Add rather than modify when you need to perform extensive changes to the delivered system.

If you are utilizing PeopleSoft-delivered fields, don't change their properties. Changing field properties may affect all other objects where the field is used. If you cannot find existing fields with characteristics that you need, you are better off creating new ones.

Rule 4. Do not change properties of PeopleSoft-delivered fields.

Avoid moving fields around in panels just for cosmetic reasons. Even if you just click on a panel field and move it inadvertently and then save the panel, the system considers this a change, and reports it during the upgrade, thereby adding to the upgrade effort.

Rule 5. Do not move fields in delivered panels just for cosmetic reasons.

The next and final rule is simple and obvious.

Rule 6. Never delete any fields from delivered records or panels. You can always use PeopleCode to hide fields in panels, if necessary.

- document every change
- use prefix letters to identify your custom objects
- add rather than modify when you need to perform extensive changes to the delivered system
- do not move fields in delivered panels just for cosmetic reasons
- never delete any fields from delivered records or panels

20.3 IDENTIFYING OBJECTS FOR CUSTOMIZATION

PeopleSoft allows you to modify the existing system. The important questions to ask are "what are the objects that have to be modified?" and "what is the best way to perform the modifications?" As we have already discussed in the previous chapter, you should always keep upgrade considerations in mind. At the same time, you should think of other implications, such as development time, ease of maintenance, possible

impact on response time, panel design constraints, coordination with other subsystems, and so on.

In order to customize a PeopleSoft-delivered application, you need to identify all objects that will be impacted by your changes. Since in real life there are usually several approaches to the same task, we will present different methods of customization in the next chapters, discussing the pros and cons of each approach.

You've already learned that a simple request to add a field to a panel may not be as simple as it appears at first glance. Therefore, when you are getting a request or even a simple question about what will be involved in the customization, do not rush to reply. Gather requirements and assess the situation by looking at the objects involved. Consider all the alternatives and select the most appropriate one. Depending on the method you selected for your customization, there may be one object or a multitude of objects that have to be customized.

Also, while deciding on the best way of customization, do not forget about the major constraint in PeopleSoft panel development: you cannot have multiple records within the same scroll bar. The only exceptions are fields from Derived/Work records and the related display fields. Based on your specific requirements, you may need to add another panel to your existing panel group.

In subsequent chapters, we will present examples of the most frequently used customizations. We'll not only discuss those examples, we'll customize the delivered PeopleSoft application using real life situations.

20.4 PERFORMING AN UPGRADE

When performing customizations to PeopleSoft-delivered applications, knowledge of the upgrade process is crucial. A good understanding of the long-term consequences of a particular change to the delivered system puts you in a better position to make a more intelligent decision on how customizations should be performed.

Let's highlight some important upgrade concepts and demonstrate them on simple examples. (Please refer to the PeopleSoft-delivered technical documentation and to the upgrade instructions when performing an upgrade process.)

First, to avoid any confusion in the upgrade's terminology, let's take a look at the PeopleSoft's definitions of different types of upgrades.

PeopleSoft categorizes upgrades into three types: *PeopleTools upgrade*, *Application upgrade*, and *Customization upgrade*.

During the PeopleTools upgrade, you move to a new PeopleTools release. This type upgrade requires installing new software and usually involves upgrading PeopleTools database objects. PeopleSoft provides database scripts to perform this type upgrade, which also involves copying new executables and dynamic link libraries delivered by PeopleSoft.

During the Application upgrade, you move to a new PeopleSoft application release. It can either be a minor application release upgrade or a major application release

upgrade. Periodically, PeopleSoft delivers application updates and fixes that you'll need to apply to your database. You use the Data Mover tool to import update projects into a stand-alone, application update database (AUDB). After that, you copy the objects into your database using Application Designer. PeopleSoft always provides the documentation (associated with a particular fix or update) that you need to follow. For the latest information on updates and fixes for PeopleSoft products, you should check the updates and fixes database in Customer Connection at www.peoplesoft.com.

When you need to migrate your newly developed or customized PeopleSoft objects from one database to another (for example, from development to production), you are performing the Customization type of upgrade within the same release level.

The aforementioned upgrades are performed differently, depending on the type of upgrade (PeopleTools, Application, or Customization) and the level of your current and the future releases.

Usually, however, you'll go through the following steps while performing any type of upgrade:

- *Populate a project* As discussed in part 2 of this book, a project is a set of records, panels, fields, and other objects grouped together to help you in application development, customization, and upgrade. A project is populated either manually while performing a customization or automatically by the Upgrade Compare and Report process.
- *Perform a comparison* Execute the Compare and Report process. Depending on the Compare Type you select, this process either compares objects in your project to the corresponding objects in your target database or compares all objects in your source and target databases and repopulates your project.
- *Change or verify the upgrade settings* The system assigns the default settings for each object in your project for the source and target databases based on the result of comparison. You can specify whether or not to upgrade each object by changing the object's Upgrade flag.
- *Perform a copy* Execute the Upgrade Copy process to copy objects from the source database to the target database or delete objects in the target database. Only objects with the Upgrade flag set to `On` are added, replaced, or deleted in the target database.
- *Execute any* `Alter/Create` *scripts as necessary* If your project contains any records that are specified as SQL tables or views you should execute the proper SQL scripts to synchronize the underlying database structure with PeopleTools records and index definitions.
- *Stamp the target database* The target database is usually "stamped" to indicate that it has changed from its previous release level. When upgrading to a new PeopleSoft release, this step is required. The target database should be stamped with the PeopleSoft release level, specified in the upgrade instructions.

Let's illustrate the Customization type of upgrade using a simple example of modifying a record definition:

Suppose we need to modify our custom MY_COMPANY_TBL record by adding a custom field, MY_COMP_BUSINESS, to it. Let's assume that this record was created as a child to COMPANY_TBL some time ago and already resides in the development, test, and production databases.

First, we create a new custom field (figure 20.4).

Navigation: GO →Application Designer →File →New →Field

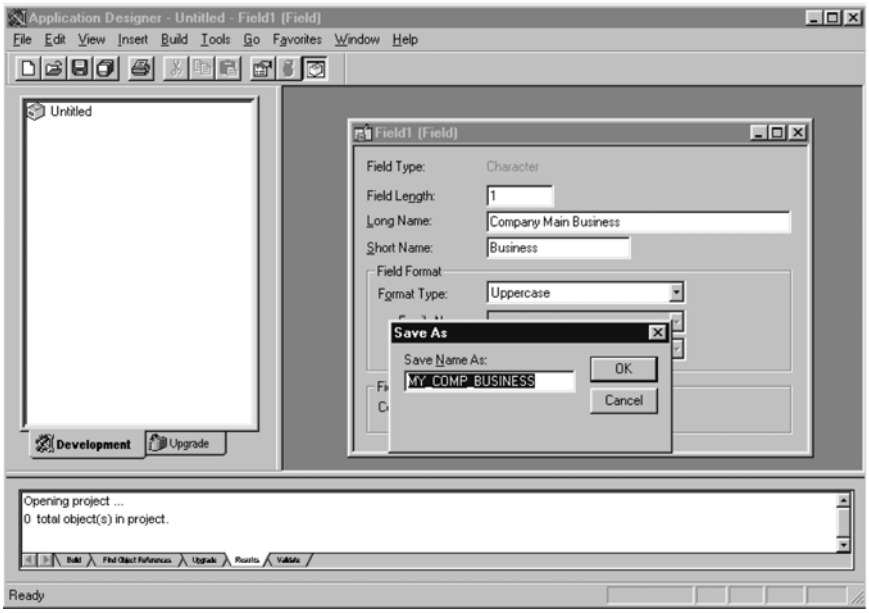


Figure 20.4 Creating a custom field MY_COMP_BUSINESS

After saving the field, let's add the new field to a new project by pressing the F7 function key or selecting Insert →Insert Current Object into Project from the Application Designer tool bar menu (figure 20.5).

Navigation: Insert →Insert Current Object into Project

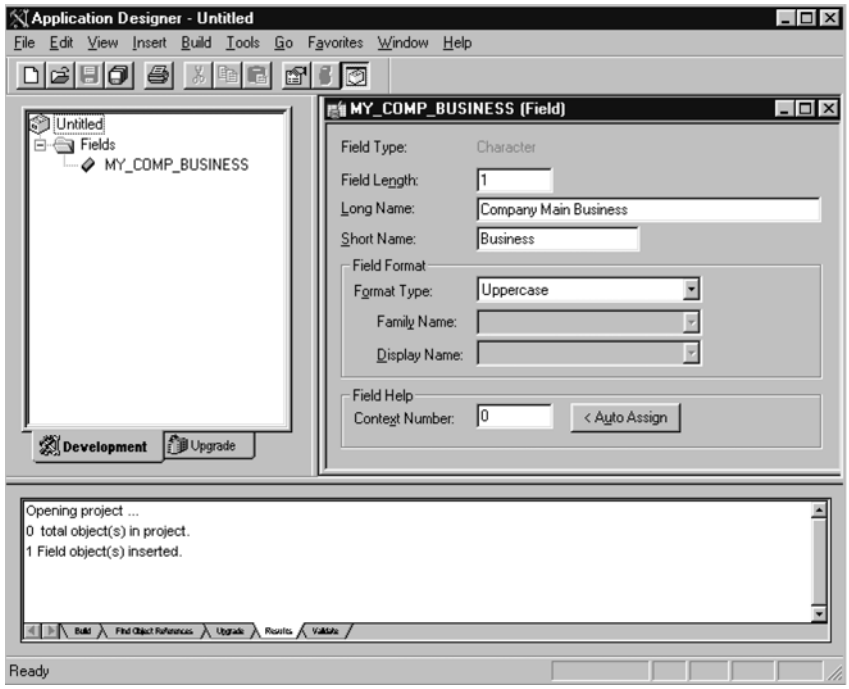


Figure 20.5 Adding a new object to a project.

Our next step is to add this field to the MY_COMPANY_TBL record definition. From the Application Designer menu, select File → Open → Record and type MY_COMPANY_TBL. After the record is displayed, highlight the field after which you want the new field to be inserted, then select Insert →Field →MY_COMP_BUSINESS. Save the record definition and add the modified object to our project by pressing the F7 function key (figure 20.6).

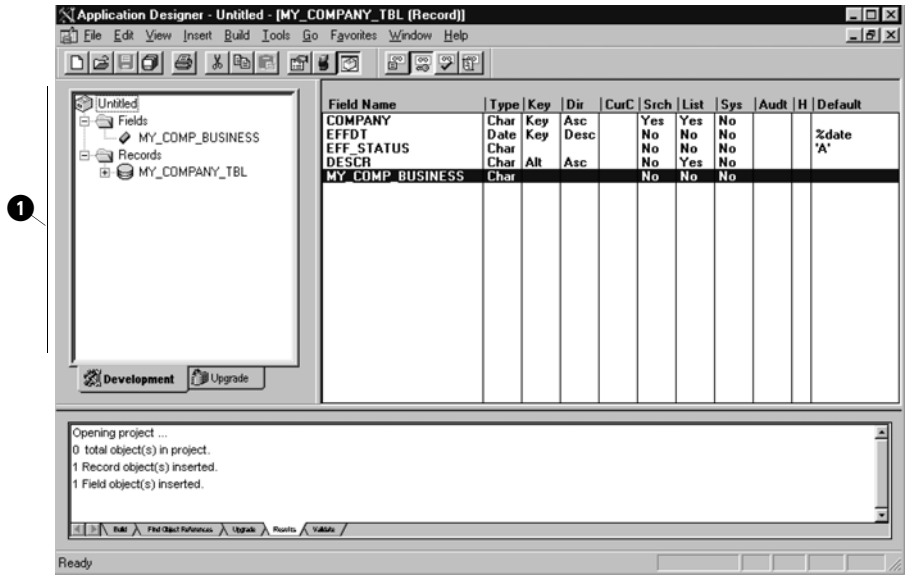


Figure 20.6 Adding the modified table to a project

As you can see from figure 20.6 ❶, all our modified objects are listed in the Application Project workspace. Let's save this project as MY_COMPANY_CHG by selecting File →Save Project As.

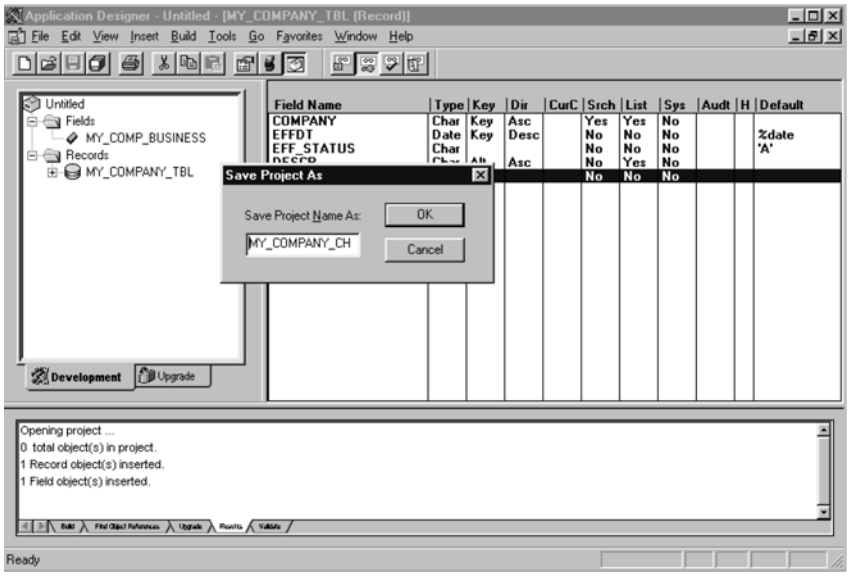


Figure 20.7 Saving our new project

Two tabs exist in the Application Designer project panel: the Development tab and the Upgrade tab. These allow you to work with the project in two different modes. The Development tab helps you perform operations on objects listed in the project. It allows you to see the object's dependencies and lists all the objects by their type. You can simply double-click on an object in the project workspace to bring the object up for any further modifications or review.

The Upgrade tab displays all objects available for upgrade from one database (source) to another (target). Let's switch from the Development tab to the Upgrade tab in our project and double-click on the field folder in the Project workspace (figure 20.8.)

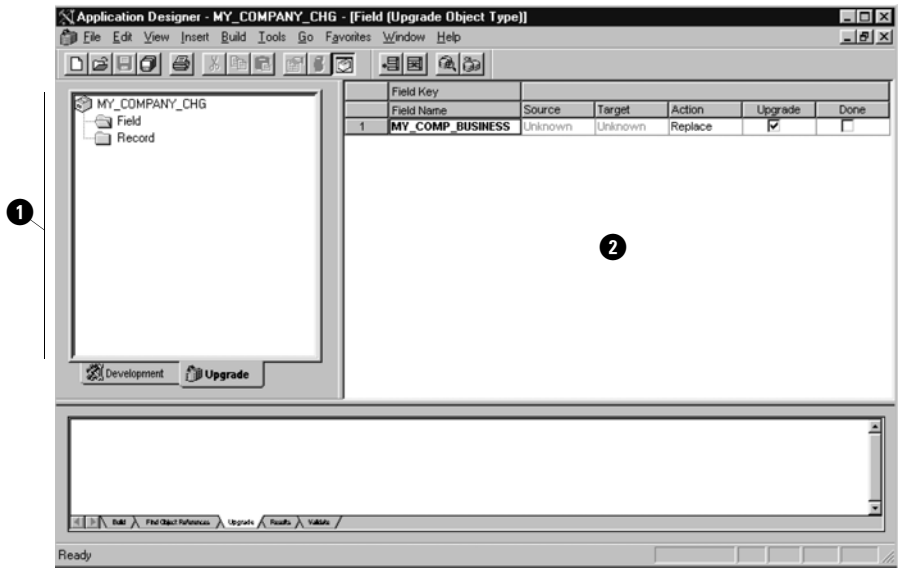


Figure 20.8 Reviewing a project in the Upgrade mode. The MY_COMP_BUSINESS field default upgrade options.

1. Project Workspace. The Upgrade View.
2. The Upgrade Definition window.

The Upgrade Definition window, which appears in the object workspace, displays the upgrade options available for the MY_COMP_BUSINESS field. The options displayed for this particular object are just the default options that have been initially set by the system automatically. In our example, we have two objects that have to be migrated to the target database, the MY_COMP_BUSINESS field and the MY_COMPANY_TBL record. The MY_COMP_BUSINESS field is a custom object

that we just created. Therefore, it is safe to copy this object to the target database without fear of overlaying any existing objects.

Another object, the MY_COMPANY_TBL record, is an example of an existing object modification. Let's double-click on the record and display the record's upgrade default options (figure 20.9).

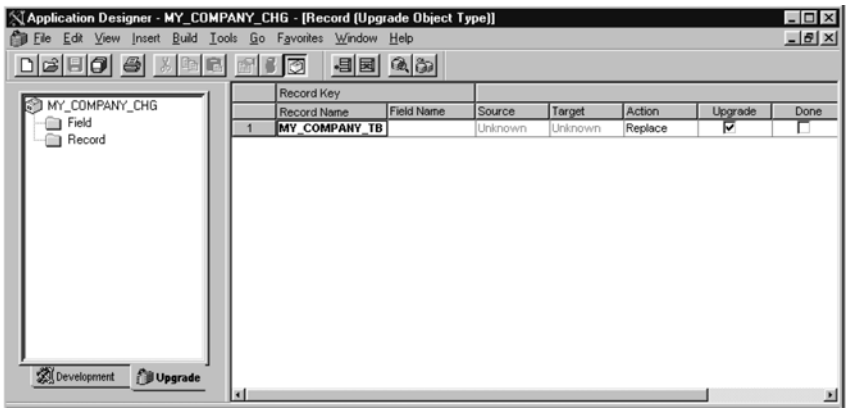


Figure 20.9 The MY_COMPANY_TBL record default options before the comparison

Using this simple example of customizing a record definition, we can demonstrate all the steps involved in the customization type of upgrade process.

Usually, all customizations and initial testing are done in the development database. The next step is the migration of modified objects to the test database for more thorough testing. And the last step is the migration of the project that includes all the changes to the production database. A migration of modified objects from one database to another is considered a Customization type of upgrade.

During the Customization Upgrade, you usually populate a project with the modified objects and copy the objects from your project to the test and production databases. Before copying the modified objects, you can execute the Upgrade Compare and Report process to compare all objects in your project with the corresponding objects in the target database.

Please note that, when executing a Customization Upgrade, it is not always necessary to execute the Compare and Report process. We can run the Compare process for our record modifications just to verify if this table has been modified by a concurrent development while we were testing it in the test database. However, if you have a strict mechanism for locking the objects before any customization, (which is always advisable), it may be safe to execute the Upgrade Copy process right away.

Let's run the comparison process (figure 20.10).

Navigation: **Tools →Upgrade →Compare and Report**

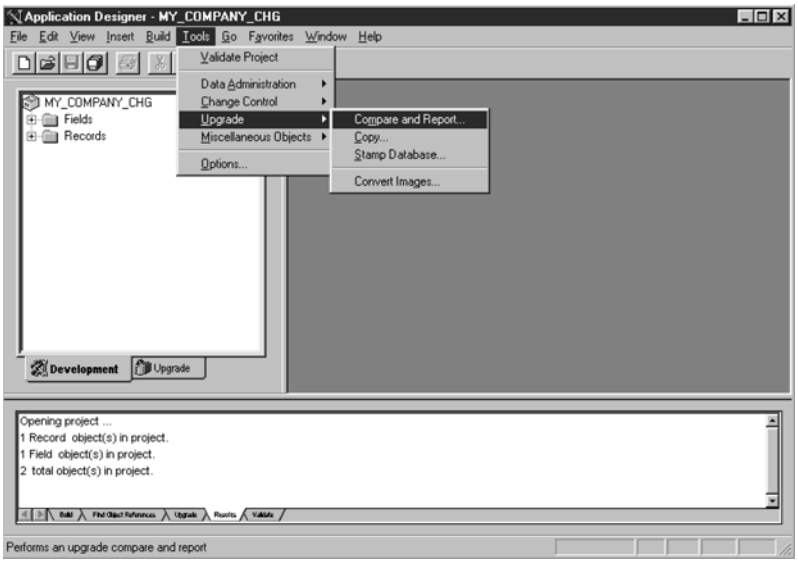


Figure 20.10 Executing the Upgrade Compare and Report process

The system asks us to sign on to the target database (figure 20.11).

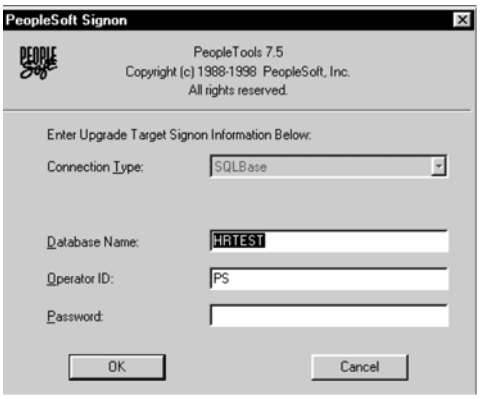


Figure 20.11
Log in to the Target database

After connecting to our test database, we are presented with the Compare and Report panel, which is a key panel in the upgrade process (figure 20.12). You can find a multitude of parameters in this panel that we will explain in more detail.

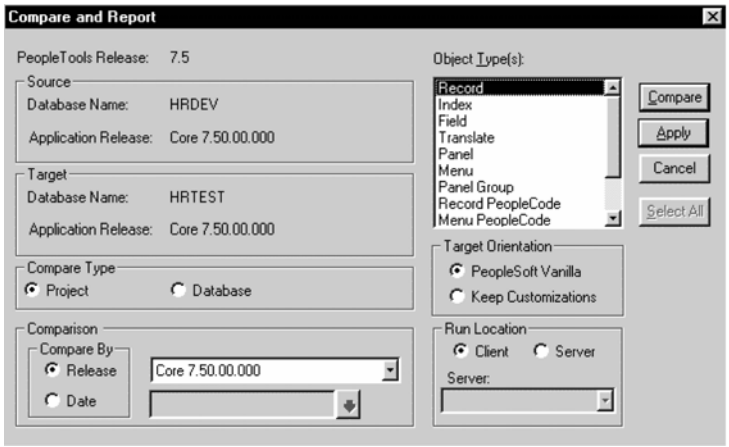


Figure 20.12 The Compare and Report panel

In the panel shown in figure 20.12, we see all parameters required for our Compare process. The first thing we need to verify is if the source and the target databases are specified correctly. In our example, we are comparing the objects in the HRDEV database to the objects in the HRTEST database.

The Compare Type can be either *Project* or *Database*. If a Project comparison is selected, only the objects in the current project are compared (of the types specified in the Object Type(s) box). The contents of the project do not change.

When a Database comparison is chosen, all objects selected in the Object Type(s) box are compared. Unlike the Project type comparison, if you choose the Database comparison, the contents of the current project are replaced with objects found during the comparison.

You can specify if you want to compare either by the release common to your source and target databases or by a particular date. The comparison process labels objects as `Changed` or `Custom/Changed` if they've been changed since the Date/Time stamp for that release level or since the date that you specify.

Depending on where you execute your comparison process, on Client or Server, you can select either one or multiple types of objects to compare. If you execute the process on Client, you can only select one object type at a time, due to locking constraints.

TIP Execute your Comparison process on Server (set Run Location to `Server`) in order to compare multiple objects at the same time. In this case, you should select more than one object type or select all from the Object Type(s) group box .

If you chose to select all object types, you can deselect any unwanted object types pressing the CTRL key and using the left mouse pointer simultaneously.

TIP PeopleSoft recommends that you execute the Menu PeopleCode and Record PeopleCode comparisons either before or after running comparisons for other object types. PeopleCode compare can be executed only on the client.

In our example, we execute the `Upgrade` type of Customization and compare only one object type, the record. Therefore, we can run our process on the Client.

The *target orientation* allows you to select either the PeopleSoft Vanilla orientation or Keep Customizations option. The target orientation tells the system how to set the upgrade checkboxes in the Upgrade Definition window for objects that were last modified by the customer in one database and last modified by PeopleSoft in the other database. If you select the PeopleSoft Vanilla orientation, the upgrade checkboxes in the Upgrade Definition window will be set to preserve PeopleSoft's changes. If you select the Keep Customizations option, the checkboxes will be set to preserve your changes. We'll talk more about the target orientation later in this chapter.

TIP During the major PeopleSoft upgrade, set your target orientation to PeopleSoft Vanilla unless your system is highly customized.

Let's now execute the Compare and Report process by clicking on the Compare button. The Upgrade Compare and Report process is initiated. You can verify the status of the job on your Process Monitor screen. The process name for the Record type of compare that we selected is `UPGCREC`.

NOTE As soon as the process is successfully executed, your project is automatically saved and closed.

If you open your project again in the Upgrade mode and double-click on the record in the project workspace window, you can see the online comparison messages in the lower part of the panel in your output window (figure 20.13). You can also print the messages by right-clicking in the output window and selecting the Print option.

Are you surprised to see two lines in our Upgrade Definition window after the record comparison is executed? The Compare process recognized that the MY_COMP_BUSINESS field is absent in the target database and repopulated our project with the new MY_COMPANY_TBL. MY_COMP_BUSINESS record field. Note that this is the only situation in which the Compare and Report process will repopulate a project during a Project type comparison.

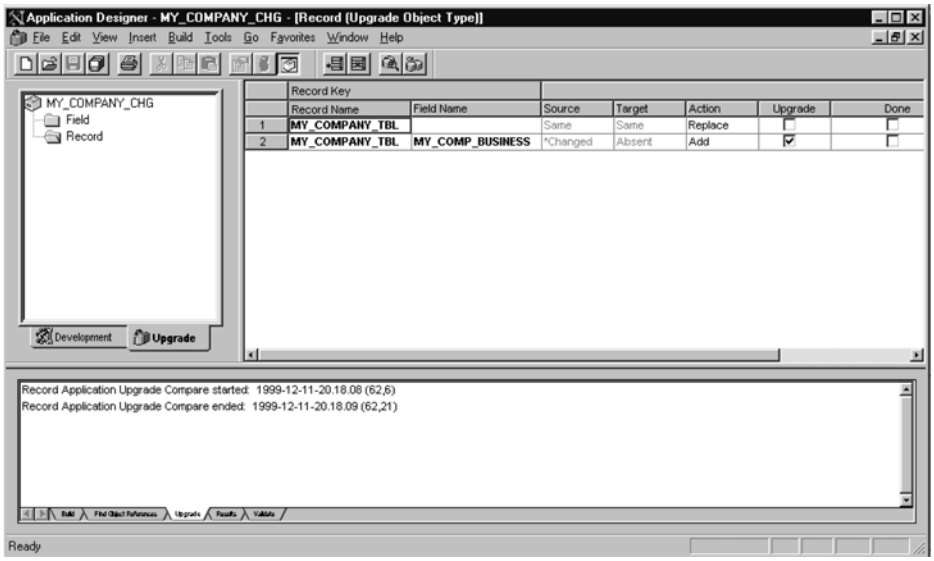


Figure 20.13 Viewing the results of the Record Comparison

NOTE When records are compared either during a database or a project comparison, any differences found in record fields will be populated into the project.

Why did the process turn on the Upgrade flag only for the second line in our project? We'll explain this in the next subsection.

20.4.1 Understanding how the Upgrade Compare process works

In the Upgrade Compare process, PeopleSoft first compares the object definition in the source database to the object definition of the target. If it recognizes a difference, it checks to see if either of these objects had changed since the comparison release.

PeopleSoft tracks the object's Date/Time stamp (`LASTUPDDTTM`) value stored in the PeopleTools System Catalog tables. For example, it stores this value in the PSRECFIELD and PSDBFIELD tables for our custom field MY_COMP_BUSINESS. Since we added a new field, MY_COMP_BUSINESS, to these catalog tables, the compare program identifies the change and marks this field as Custom Changed in our source database and as `Absent` in our target database.

Another system catalog table, the PSRECDEFN table, contains the Date/Time stamp equal to the last time we modified the MY_COMPANY_TBL record definition. It is stamped by the system only if any of the record properties (such as the record description, a query search record, a parent record, and so forth.) are modified. Since, in our example, we have not modified any of the record properties, the system is not

going to update the PSRECDEFN table in the target database and, therefore, the first line in figure 20.13 has not been marked for upgrade.

Please refer to appendix C of this book for a list of the PeopleSoft System Catalog tables. A familiarity with these tables will definitely help you to understand the PeopleSoft system from inside and will no doubt make you a better developer.

Another important component of the compare process is the release date/time value for the comparison release level. This value, `RELEASEDTTM`, is stored in another PeopleSoft table, PSRELEASE.

PeopleSoft then compares the date our object was last modified with the release date. If the date of the object is greater than the release date, PeopleSoft considers this object changed.

In addition to the date comparison, the system checks the last `operator ID` that modified the object. It then considers the object changed if it was modified by someone other than PeopleSoft (`LASTUPDOPRID <> 'PPLSOFT'`).

During the Comparison Upgrade process, the system determines `Status`, `Action`, and `Upgrade` values for each object. The `Status` value is defined for both the source and target objects.

`Status` may have the following values: `Unknown`, `Absent`, `Changed`, `Unchanged`, `Changed *` , `Unchanged *`, `Same`.

As you can see from figure 20.9, the status values for our source and target databases are `Unknown`. This is a default status, and it means that the object has not been compared. This is also a temporary status, assigned when an object is manually inserted into a project. As soon as the compare process is executed, this status is replaced with the appropriate status.

The `Absent` status value means that the object was found in the other database, but not in this one. In our example, as you can see in figure 20.13, after we ran the compare process, the status value of `Absent` is specified for the target database.

The status value `Changed/Unchanged` means that the object was changed/unchanged by PeopleSoft (PSOFT user) since the last comparison release.

The status value `Changed *` means that the object was modified by someone other than PeopleSoft (the `LASTUPDOPRID` value is not `PPLSOFT`) since the last comparison release. That is exactly why we can see the value of `Changed*` in figure 20.13 for our source database modified object.

The status value `Unchanged *` means that the object was modified by the customer (LASTUPDOPRID is not PPLSOFT) prior to the comparison release.

Finally, the status value `Same` means that the object has not been modified. This status appears only as a result of a `Project` type of compare and not a `Database` type of compare.

When upgrading to a new PeopleSoft release, all custom objects developed by users should have the `Absent` status in the source (new release) database. On the other hand, all of the new objects developed by PeopleSoft should have the `Absent` status in the target (your Production) database.

The Upgrade process assigns one of the following Actions to each object: `Add`, `Delete`, or `Replace`, based on the comparison process.

Action `Add` means that the object will be added to the target database. Action `Delete` means that the object will be deleted from the target database. Action `Replace` means that the object in the target database will be replaced with the corresponding object in the source.

In our example, the record field object MY_COMP_BUSINESS has `Action = Add`, because this field will be added to the target database.

You would be able to decide whether to execute the Action during the Upgrade Copy process, which is the actual migration. Take a look at the next value, the `Upgrade`. This value could be set to `Y` or `N`, which indicates to the Copy process whether the corresponding Action should or should not be executed. During the new release upgrade, it is the upgrade team's responsibility to make certain that all the values are set correctly. The system can only set the `Upgrade` values to `Y` or `N` based on the comparison results and the target orientation. The target orientation allows the user to either choose the upgrade to keep PeopleSoft changes or retain custom changes in the target database.

Suppose the new release came with some modifications to the PERSONAL_DATA record. Let's assume that we also modified this record. During the Upgrade Compare and Report process, the status of this object will be `Changed` in the source database (the PeopleSoft's new release database) and `Changed *` in the target database. The Action will be `Replace`, and the Upgrade value will be set to `Y` or to `N`, depending on the orientation that is selected for our upgrade. If we choose to take PeopleSoft's objects when both have been changed, then our modifications will be overwritten. If we choose to keep the target unchanged, then PeopleSoft's changes will be dropped. In cases such as this, the customizations have to be examined, and the decision made on whether to keep the customizations, abandon them, or merge with the new PeopleSoft-delivered modifications. Each case should be looked up separately in situations similar to the one described.

Let's get back now to our Compare process. Since we execute our process on the Client, we should perform the compare for each object type separately. Our Upgrade project contains two objects: a record and a field. We just executed the record comparison. Should we now compare the field? In this particular case, it is not necessary since we created a new field, and we know that this field does not exist in the target database. We can go directly to the Copying process.

20.4.2 Copying a project to the target database

When you have completed all your upgrade settings, the next step is to copy the project into your target database. It is a good idea to check the `Change Control` locking status of your target database and check your copy options before initiating a Copy process. (Please refer to PeopleSoft technical documentation for information about Locking and Upgrades.)

Let's select File →Project Properties to verify our Copy options.

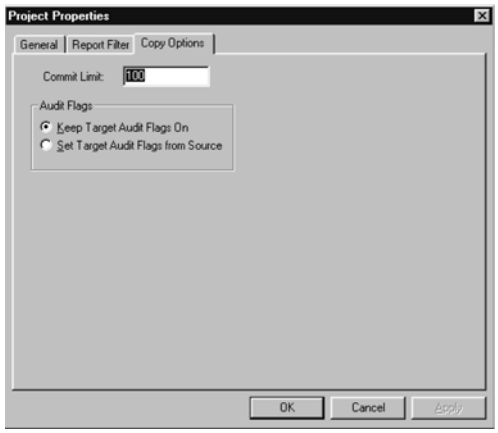


Figure 20.14
Verifying the project's Copy Options

As you can see from figure 20.14, the default `commit limit` is set to `100`. You can modify this number based on the amount of time it takes you to complete the process. Increasing this number speeds up your process. Be careful: if something goes wrong, it may increase the amount of work in your recovery process. Always consult your Database Administrator before you change the `commit limit`.

In the Audit Flags box, you specify the Audit Flags setting to either `Keep Target Audit Flags On` or `Set Target Audit Flags from Source`. This allows you to preserve your target database audit (PSAUDIT) settings if you choose the default option or to bring flags from the source database to the target.

For our task, let's leave all the default options on, click on the Cancel button, and initiate a Copy process.

The system asks you to sign on to the target database. Let's sign on to our HRTEST database and click on the OK button.

The system displays the Copy dialog panel (figure 20.15).

Navigation: Tools →Upgrade →Copy

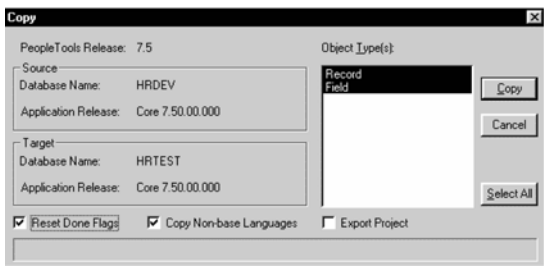


Figure 20.15 The Upgrade Copy dialog panel

TIP Always verify if your source and target databases are specified correctly before you execute a Copy process.

If you have the option of `Reset Done Flags` turned on, the system will reset all done flags for selected objects before performing a copy. If you have not selected `Reset Done Flags`, it will only copy the objects with Done flags turned off.

When Copy Non-base Languages option is on, the system copies all base and related language objects to the target database. Otherwise, it copies only base language version tables.

You can also check the Export Project box `ON` to copy the current project to your target database before copying any other objects.

Now we can perform the actual copy process. Just click on the Copy button from the Upgrade Copy dialog panel.

When the Copy process is executed, it displays the messages on the Upgrade Project output window. You can click on each object to check the messages and to verify if the Done flag is set to `Y` as shown in figure 20.16.

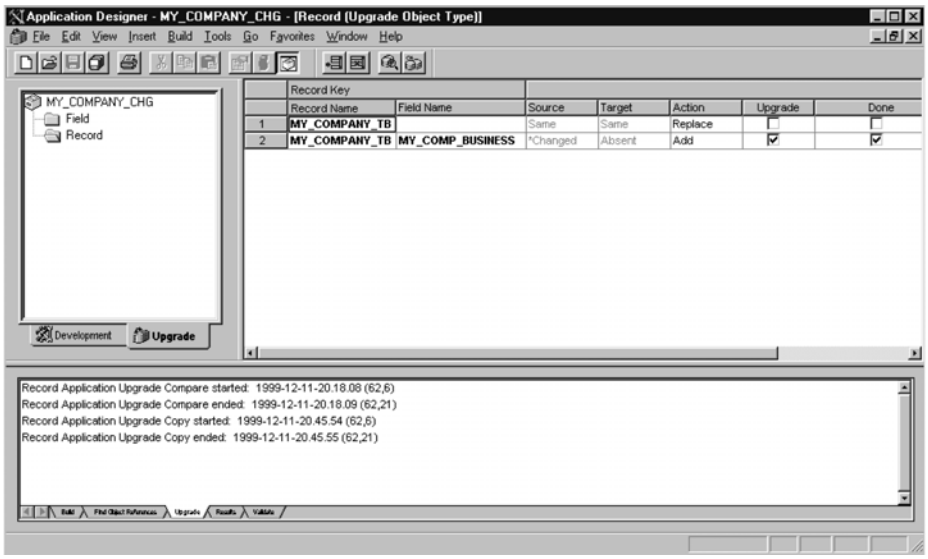


Figure 20.16 Verifying the results of the Upgrade Copy process

Just as with the panel shown in figure 20.16, if you click on field in your project workspace, you see that the Done flag is `ON` for field copy as well.

20.4.3 Executing Alter/Create scripts

Since we modified the record definition in the target database, we should create and execute the SQL necessary to synchronize the underlying database structure with the PeopleTools record.

Navigation: Build →Current Object

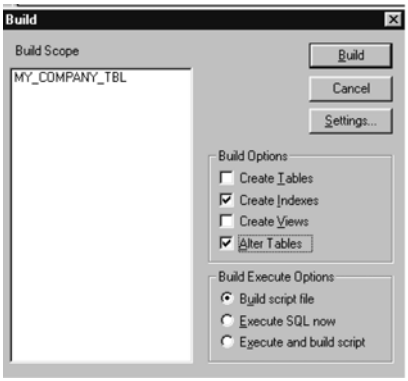


Figure 20.17 Building a script to alter the MY_COMPANY_TBL table

Let's login to the HRTEST database, open our project, and switch to the Development tab. Double-click on the MY_COMPANY_TBL table in the project workspace to display the record definition.

Since MY_COMPANY_TBL already exists in the target database, we use the Alter Tables as our Build Options, and Build Script file as Build Execute Options. The results of the Build operation are written to a script file that our Database Administrator can execute.

After MY_COMPANY_TBL is altered, we move to the next step. Our last step in the Upgrade process is to stamp the target database.

20.4.4 Stamping the database

After copying a project into the target database, you can change the customer release number to specify that it has changed from its previous customer release level. This process, called "stamping the database," helps you keep track of all customer releases to this version of your database.

This step is necessary only for PeopleTools and Application types of upgrade.

NOTE In order to stamp your database with the customer release number, you have to be logged on to this database.

Navigation: Tools →Upgrade →Stamp Database

Figure 20.18 Stamping the database

The stamping of your database is optional when you change your customer release, but it is required by PeopleSoft when upgrading to another PeopleSoft release level. It is usually included in the upgrade instructions.

KEY POINTS

1. A customization is any change to a delivered application.
2. When performing customizations, try to minimize the impact on the future system upgrades.
3. Document every change.
4. Use prefix letters to identify your custom objects.
5. Use "add" vs. "modify" approach when performing the extensive changes to the delivered PeopleSoft system.
6. Avoid changing PeopleSoft-delivered field properties and cosmetic changes in the delivered panels.
7. Do not delete any fields from the delivered panels and records.
8. PeopleSoft divides all upgrades into three types: PeopleTools Upgrade, Application Upgrade, and Customization Upgrade.
9. The following steps are usually performed in any kind of upgrade:
   - populating a project
   - performing a comparison
   - changing or verifying the upgrade settings
   - performing a copy
   - executing `Alter/Create` scripts as necessary
   - stamping the target database

CHAPTER 21

Customizing delivered panels

Adding a field to a panel is one of the most frequent requests PeopleSoft developers receive. This task can be greatly simplified if the added field already exists in one of the records attached to the panel. We'll start with the simplest example, one which PeopleSoft marketing representatives usually use to demonstrate how easy it is to customize delivered PeopleSoft applications using PeopleTools. Our goal is not only to show you how to do customizations, but to discuss different ways to perform customizations. We also want to stress the importance of thinking about the impact of a particular change on future upgrades.

We'll begin with exercise 1:

> Add a field to the Personal Profile panel to specify whether the employee belongs to the Highly Compensated Employee category.

Our objective here is to customize a delivered PeopleSoft panel to achieve the panel illustrated in figure 21.1.

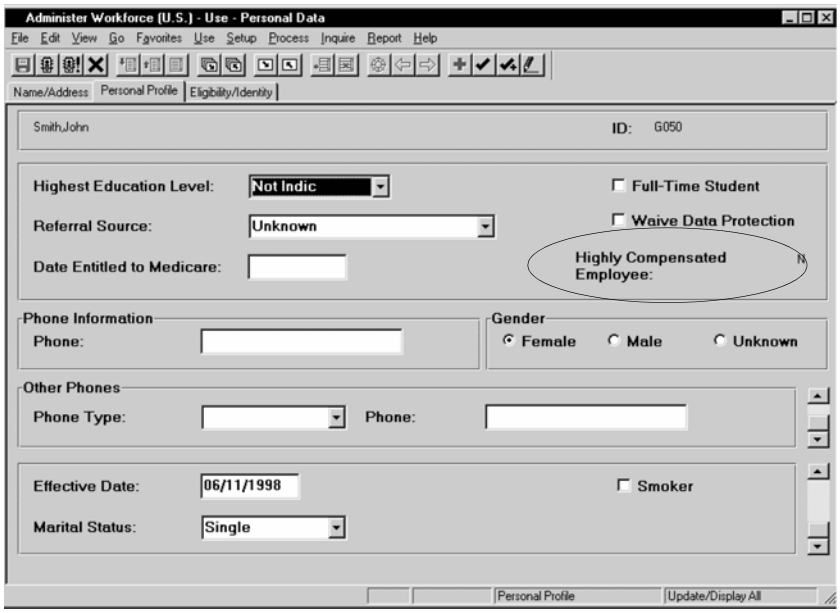


Figure 21.1 The Personal Profile panel that includes a new field

If you have some experience with PeopleSoft, you may know which of the PeopleSoft objects (fields, records, and panels) are involved in this change. PeopleSoft has so many records and fields that it's impossible to remember everything. We will discuss some handy PeopleTools techniques that can be used in order to obtain this information.

21.1 WHAT OBJECTS SHOULD BE CUSTOMIZED?

Let's do a little research here. Open a panel that your user wants to customize and find out what records are linked to this panel. Before we open the Personal Profile panel, let's also turn on the option of displaying the panel name (figure 21.2).

Now we can access our panel and find the physical object name of the Personal Profile panel.

Navigation: GO →Administer Workforce →Administer Workforce (U.S) →
Use →Personal Data.

Let's select `Update/Display` and enter a part of an employee name, for example, "Smith." The system presents you with a list of all employees whose last name starts with "Smith." Let's select "Smith, John" and bring up the Personal Profile panel for the selected employee (figure 21.3).

Navigation: View →PanelName

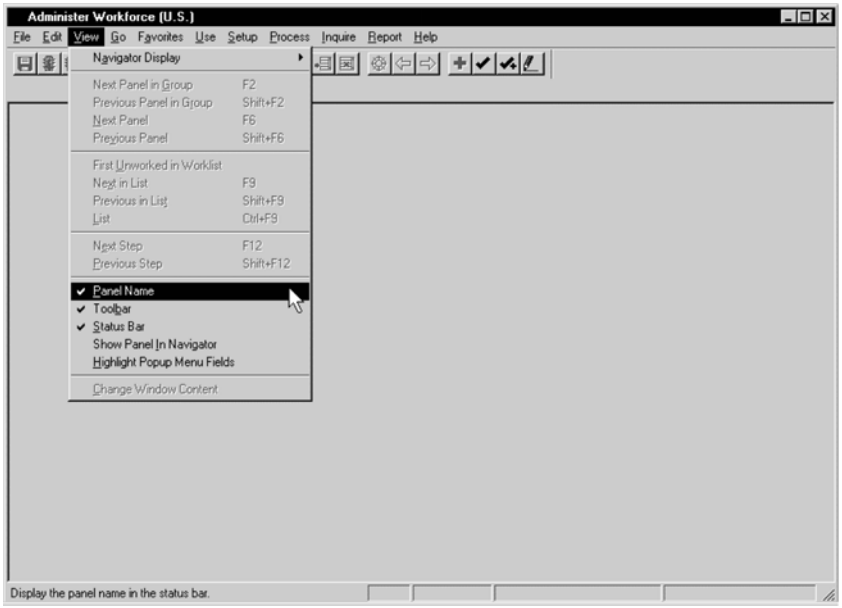


Figure 21.2 How to display the actual panel name

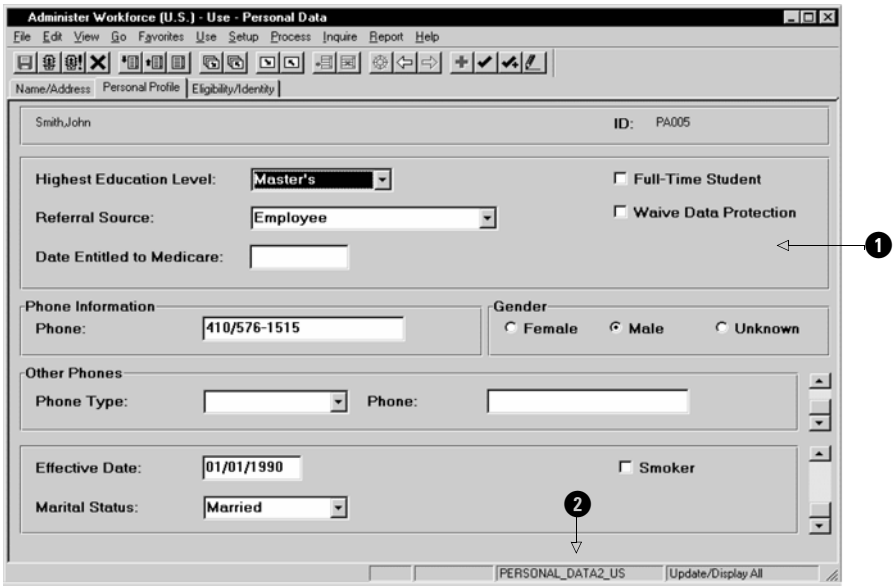


Figure 21.3 The Personal Profile panel

❶ Add Highly Compensated Employee field to this sub panel

❷ Panel Name as it is referenced in PeopleTools

As you can see in figure 21.3, the system displays the panel name at the bottom of the screen. Now that we know the name of the panel, our next step is to look at the panel via the Application Designer and find out what records are used by this panel (figure 21.4).

Navigation: Go →PeopleTools →Application Designer →File →Open →Panel → PERSONAL_DATA2_US

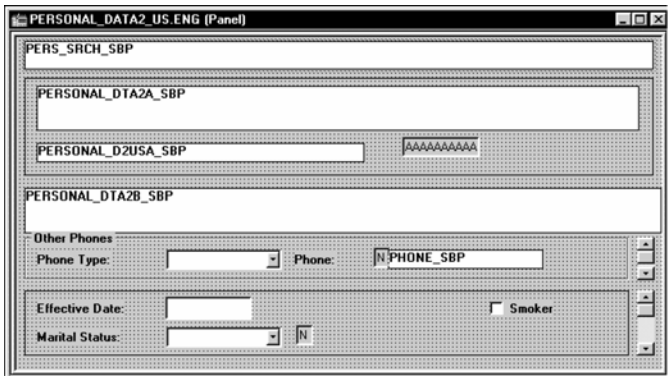


Figure 21.4 The Personal Profile panel (PERSONAL_DATA2_US) displayed via the Application Designer

To see the records and fields used in this panel, click on Layout and select Order.

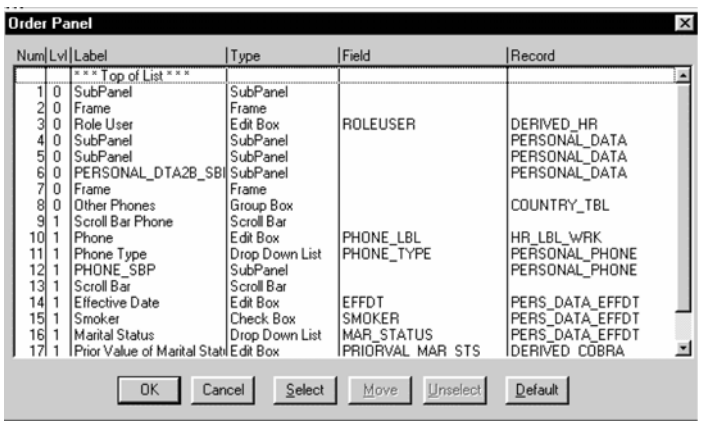


Figure 21.5 Subpanels, records, and fields that are used in the PERSONAL_DATA2_US panel

As we scroll through the panel shown in figure 21.5, we can see all the records used in the PERSONAL_DATA2-US panel and its subpanels. Let's examine records in this panel and see if the Highly Compensated Employee field belongs to any of them. The PERSONAL_DATA is one of the core records in the HRMS database. Let's open it in the Application Designer.

Navigation: **Go →PeopleTools →Application Designer →File →Open →Record → PERSONAL_DATA**

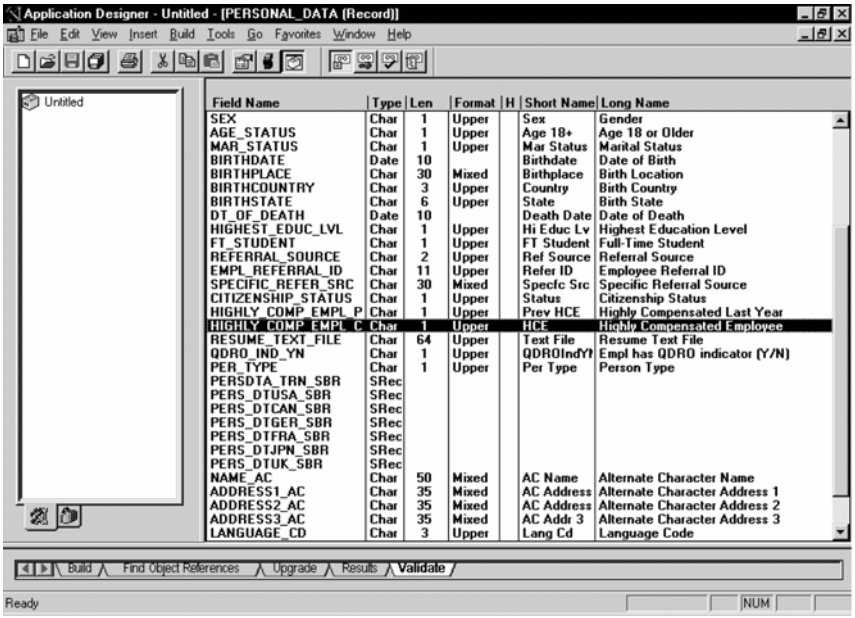


Figure 21.6 The PERSONAL_DATA table

Two fields exist which may be used for our purposes: The first is Highly Compensated for the previous and current year. After verifying requirements with our user, we decide to use the second one.

Stay in touch with your users during all stages of development. Do not wait until the end of your implementation to find the answers to your questions.

This makes our task simple. We just need to add the field to the panel. Take a look again at the panel in figure 21.4. Our users want to add a new field to the upper portion of the panel. As you can see, we need to modify the PERSONAL_DTA2A_SBP subpanel.

21.2 MODIFYING A PANEL

We already know that in order to add the required field to the Personal Data panel, we need to customize the PERSONAL_DTA2A_SBP subpanel. Let's open it and perform the actual modification.

Navigation: Go →PeopleTools →Application Designer →File →Open →Panel → PERSONAL_DTA2A_SBP

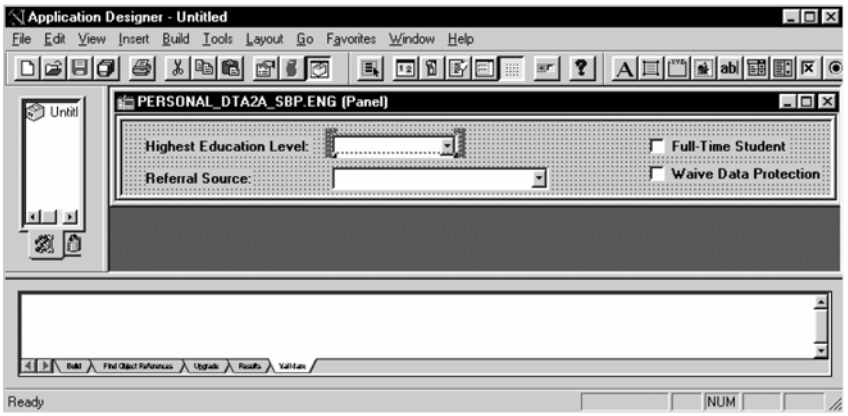


Figure 21.7 Modifying PERSONAL_DTA2A_SBP. Step 1: Opening a panel

Let's resize the subpanel a little bit and add a field to the panel by clicking on the Insert menu item and selecting an appropriate panel field type from the Application Designer tool bar. What type panel field are we going to add to our panel? As you can see in figure 21.8, different types are available: `Edit Box`, `Check Box`, `Drop Down List`, and so on.

Generally, the following guidelines are used for selecting a field type. (Please refer to part 2 of this book to learn more about panel design.)

- `Edit Box` is used for text data entry.
- `Drop Down List` is usually used to allow data selection from a list of translate value descriptions, or a prompt list.
- `Long Edit Box` is associated with long character fields from a Record definition.
- `Check Box` is used for data entry fields that can have one of two values: on or off. Y/N fields are usually the best candidates for the check boxes.
- `Radio Button` represents one value for a field with multiple defined values.

TIP Use Edit Box as a general purpose panel field type for display fields.

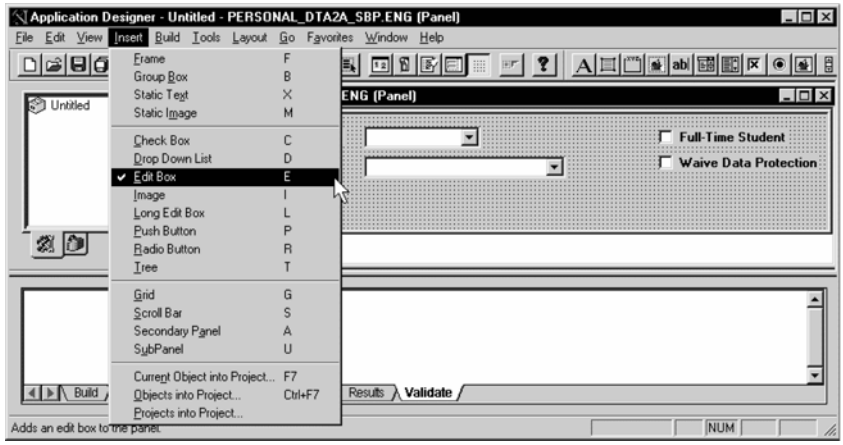


Figure 21.8 Selecting an appropriate field type

In order to select the right option, we need to see the properties of the HIGHLY_COMP_EMPL_C field. You can either open the HIGHLY_COMP_EMPL_C field from the Application Designer menu or right mouse click on the field in the record as shown in figure 21.9.

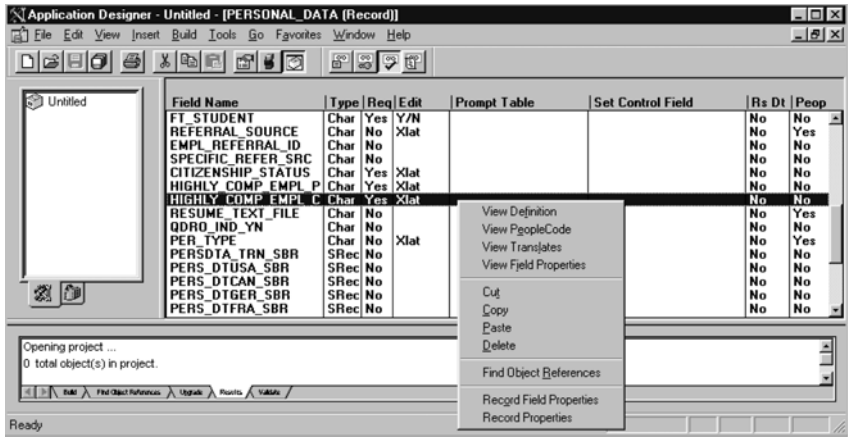


Figure 21.9 Inspecting the field's properties

From figure 21.9 we can see that our field has some translate values. Also, we have to keep in mind that users do not want to make this field a data entry field. Since they will be using it for information purposes only, we can assume that the `Edit Box` will work fine to represent this field in the panel.

Getting back to the panel in figure 21.8, let's select the `Insert/Edit Box` and place the field on our panel.

After placing the selected field on the panel (by default, the system assigned "Dummy Name" to the new field), we need to define panel field properties. Using a right mouse click on the field, select Panel Field Properties.

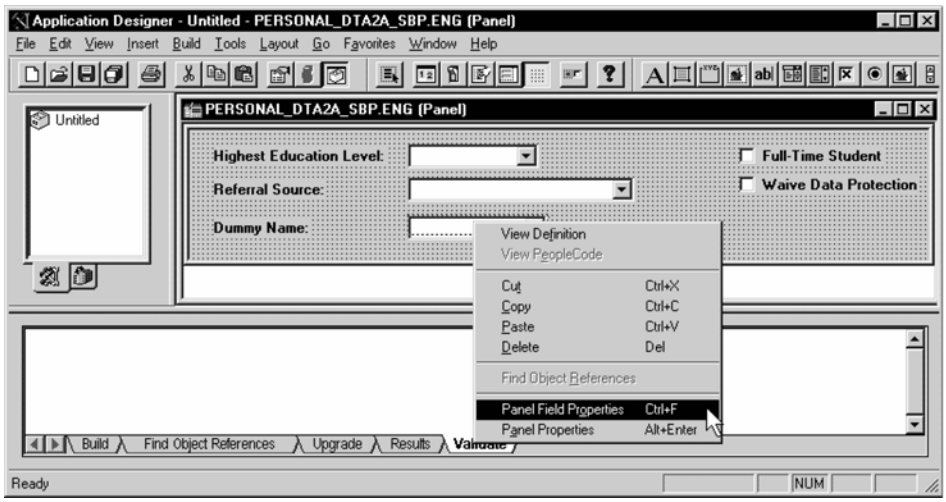


Figure 21.10 Assigning the newly added field its properties

Now we can specify the record, PERSONAL_DATA and the field, HIGHLY_COMP_EMPL_C (figure 21.11).

On the second tab of the Panel Field Properties panel, the Label tab, we need to select the field label. Keeping in mind that this panel may be used in different languages, it's always better to select the `RFT Short or Long` rather than `Text`. Why? If you specify `Text` in the label field, and the panel is using languages other than your base language, the label will still appear in your base language. When you specify the `RFT Short` or `Long` description, the label is taken from the corresponding related language table (figure 21.12).

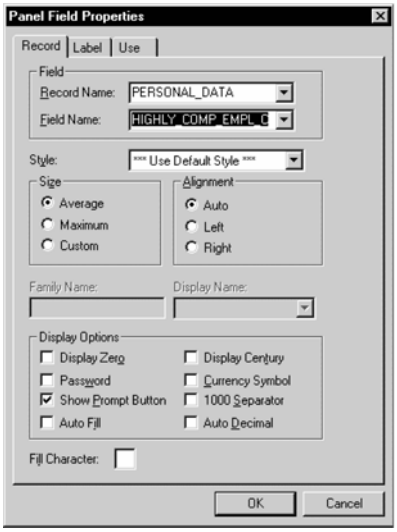


Figure 21.11 Specifying the record and the field

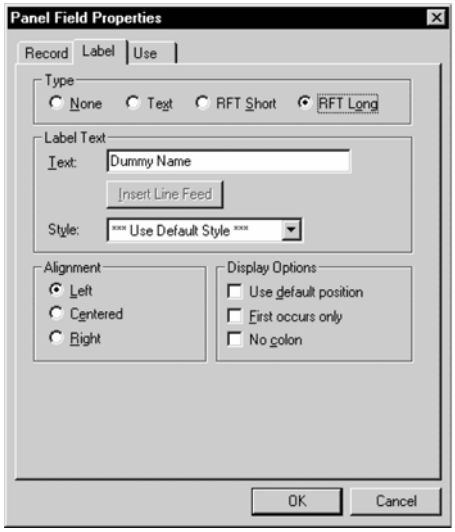


Figure 21.12 Selecting a label for our field

TIP In global development projects, always specify the `Label` type as `RFT Short` or `RFT Long`

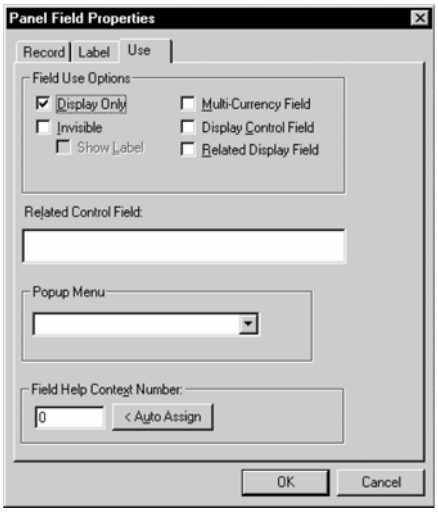


Figure 21.13 Defining our field as Display Only

On the Use tab of this panel, we mark this field as `Display Only`. This means that the field is used for information only as our users requested, since they do not plan to update this field on the panel.

Clicking on the OK button results in adding the new field to the subpanel (figure 21.14).

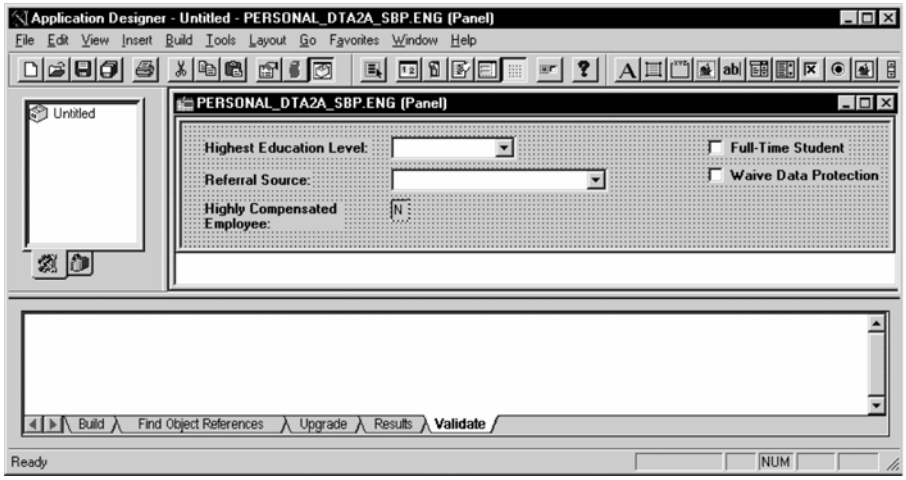


Figure 21.14 Adding the Highly Compensated Employee field to the subpanel

Save the modified panel and select File →Object Properties or ALT-ENTER to document the changes.

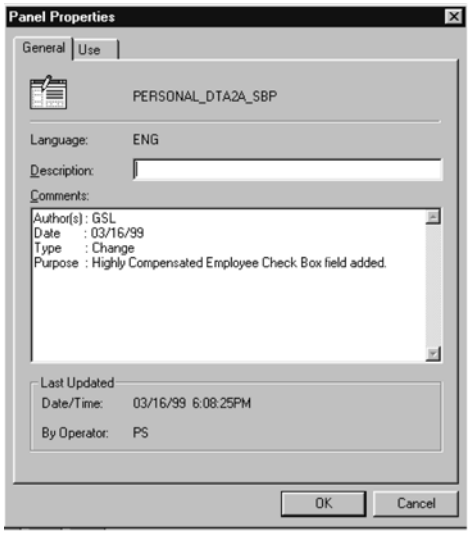


Figure 21.15 Documenting our changes on the Panel Properties

Our comments will be useful during the upgrade so we save them by clicking on the OK button. Now we can test the change.

21.3 TESTING THE MODIFICATIONS

Let's first click on the button or select Layout →Test Mode to ensure that the field really is accurately placed.

Navigation: GO →Administer Workforce →Administer Workforce (U.S) → Use →Personal Data.

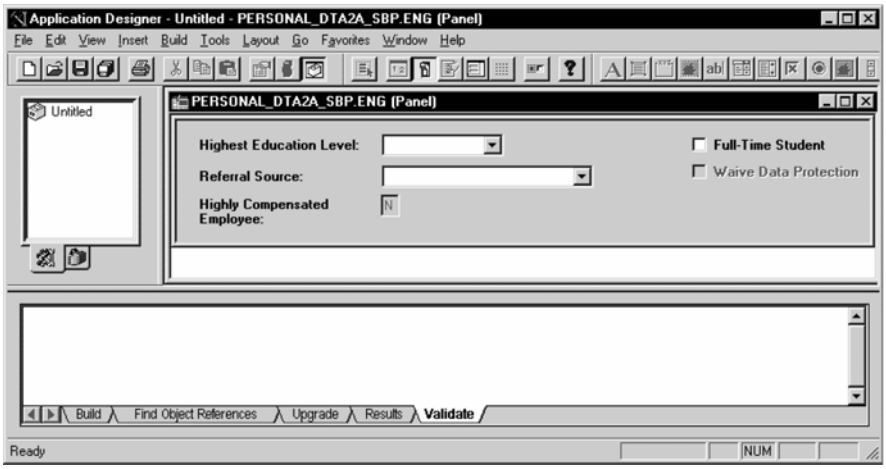


Figure 21.16 Testing the subpanel

The panel looks good in test mode. Now, we can perform a real test.

Let's select `Update/Display` and enter the same name, "Smith." After selecting an employee—for example, "John Smith"—our newly modified panel appears as shown in figure 21.17.

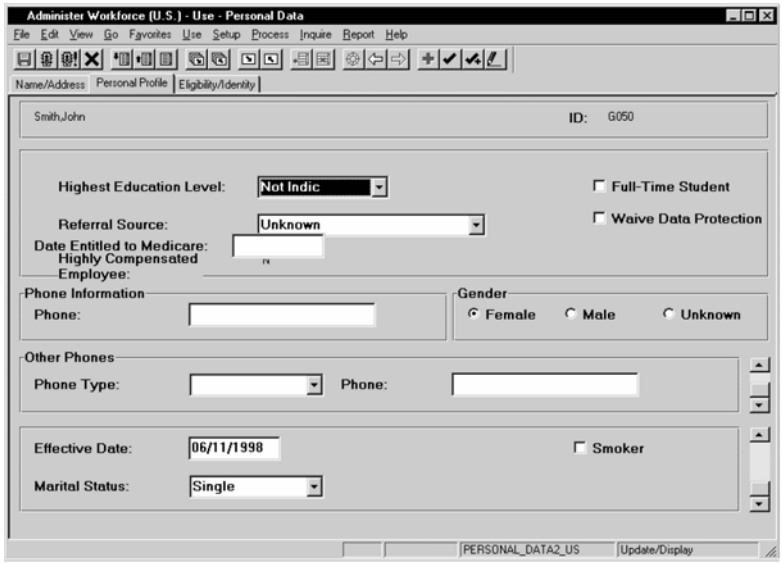


Figure 21.17 Testing the modified panel

As you can see, our new field is barely visible on the panel. It is blocked by another field. What happened? That's not what we expected, but that's exactly what testing is for! What could have possibly gone wrong? Remember that we added our new field to a subpanel and viewed the subpanel in test mode. It's obvious now that we overlaid other parts of the panel. Let's take a look at our panel again.

The panel in figure 21.18 consists of a number of subpanels. One of the subpanels, PERSONAL_D2USA_SBP, is placed over the PERSONAL_DTA2A_SBP subpanel. Also, if you compare this figure with the panel on figure 21.4, you notice that the PERSONAL_DTA2A_SBP subpanel occupies much more space now. Remember that we resized this subpanel in order to fit the new field into it but we forgot about other subpanels. This is exactly what caused our problem. Therefore, careful planning has to be done in order to find the correct placement in the panel for our new field. In our development, we were actually concentrating on finding the correct field and record. The test shows that space planning is equally important. If you display the panel (shown in figure 21.18) in test mode, you realize that the better choice is to place our field in the right corner of this subpanel where it is not blocked by other subpanels. Let's open our subpanel again and move the field to the right by dragging it to the desired position on the subpanel (figure 21.19).

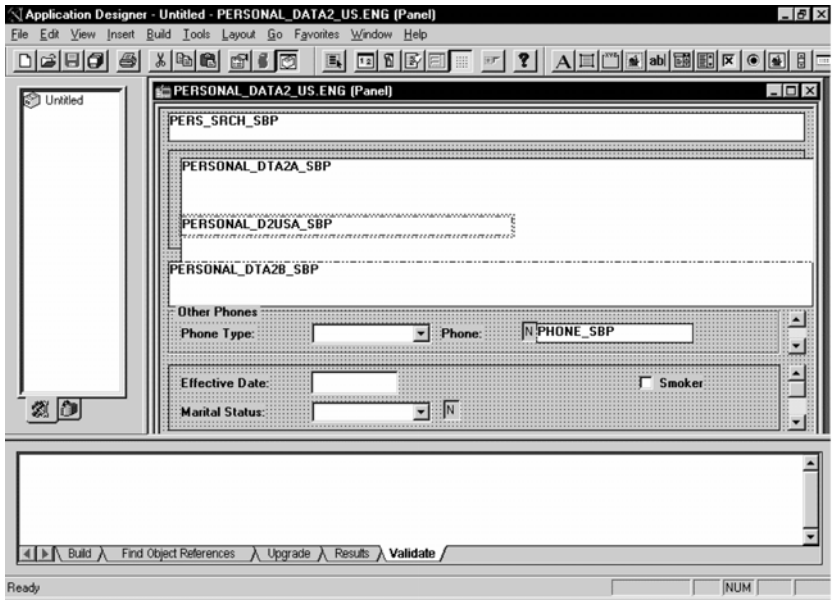


Figure 21.18 Inspecting the PERSONAL_DATA2_US panel

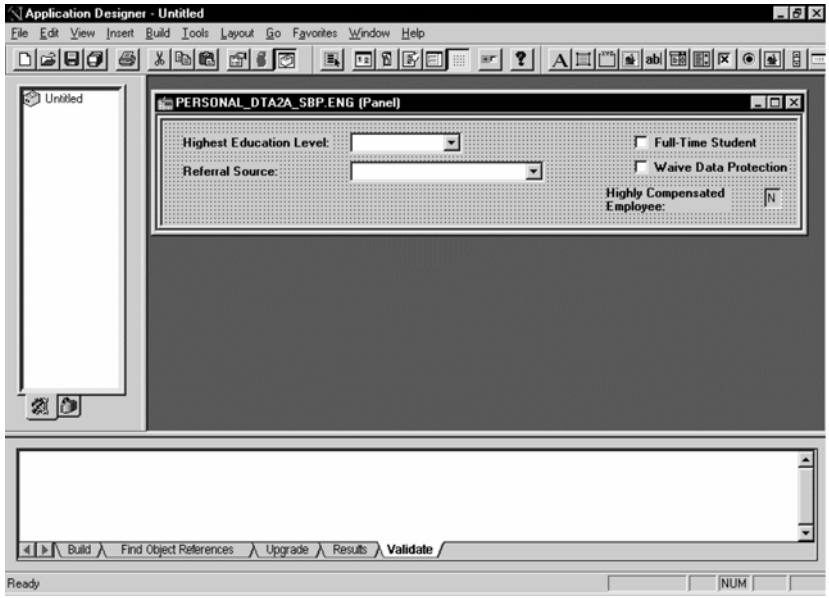


Figure 21.19 The modified PERSONAL_DTA2A_SBP subpanel

After the change to the panel is saved, let's test it again (figure 21.20).

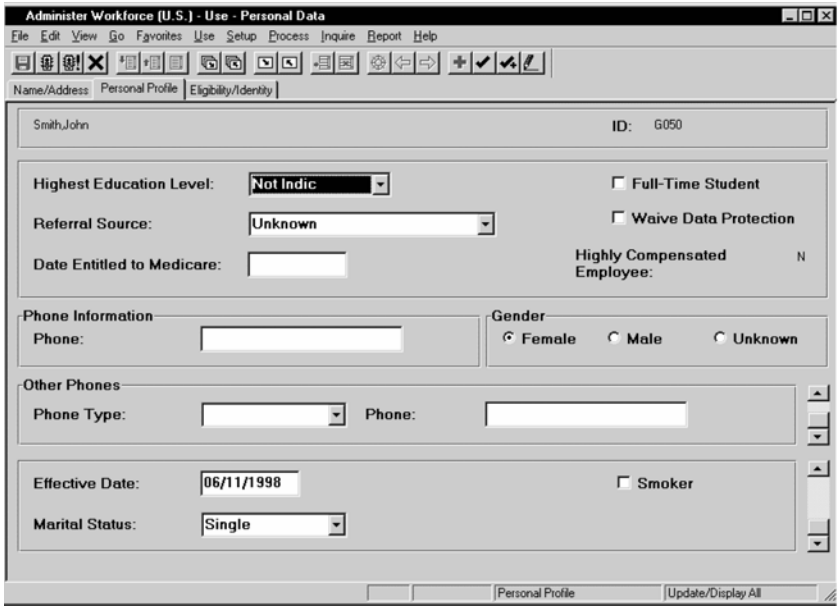


Figure 21.20 Testing the modified panel

Our new field is in place, and it looks as if it has always been there. It will help users to identify highly compensated employees as requested. We performed our modification just by adding an existing field to a subpanel. One other important point must be made: Since we added a field to the subpanel, all panels using this subpanel now display the new field. Therefore, let's first find out which other panels include the modified subpanel, then we can decide if our changes are still appropriate.

TIP Use the Find Object References utility to find all the objects that reference a modified subpanel.

In order to do that, let's open our subpanel in Application Designer and request the Object References.

As shown in figure 21.21, two more panels use our subpanel. These two panels will definitely have a new look after our modifications are performed. If our users do not want to see a new field on the other panels, we can hide this field by using a simple PeopleCode statement based on the panel name.

Navigation: Edit →Find Object References

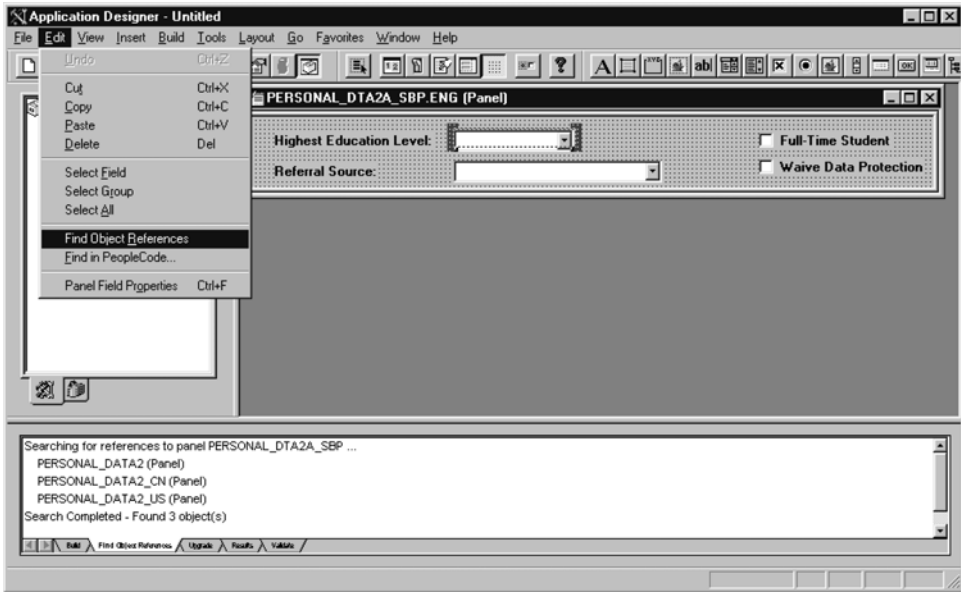


Figure 21.21 Finding all panels that use the PERSONAL_DTA2A_SBP subpanel

21.4 POSSIBLE IMPACTS ON FUTURE UPGRADES

Since our customization was simple, we chose to modify the delivered panel. Another alternative could be to copy the delivered panel under a different name and then apply all customizations. In this case, we would have to modify a panel group to replace the delivered panel with our newly customized one. There are pros and cons in each approach. In subsequent chapters, we'll demonstrate other approaches as well and let our readers decide which method of customization they prefer.

The only modified object in our customization was a delivered panel, and we documented our change in the Panel Object Properties Comment box. Will this little change really impact the new release upgrade? To answer this question, let's consider the following situations:

1. The Personal Profile panel in the new release has not been changed by use of PeopleSoft.
2. Another field has been added by PeopleSoft to the same panel in the new release.
3. The same field (Highly Compensated Employee) has been added by PeopleSoft to another panel in the same panel group.

In case 1, the Upgrade Compare and Report process identifies the change. After reviewing the modification, you can drop the panel delivered by the new PeopleSoft release, thus preserving your modified panel in your production database.

NOTE Please bear in mind that when upgrading to a new application release, the source database usually contains all PeopleSoft new release objects, while the target database is a copy of your production database.

In case 2, the Upgrade Compare and Report process tells you that both objects have actually been changed. Your task, therefore, becomes more complex, since you have to review all the changes and merge your modifications with the new PeopleSoft-delivered panel.

Case 3 is even more complex. Beside recognizing that PeopleSoft actually delivered the same or similar functionality, you have to decide which modification to leave and which one to drop. In cases like this, we recommend adapting PeopleSoft's way, even though your users may already have become accustomed to the change you delivered and may find PeopleSoft's change less convenient.

If PeopleSoft delivers the same or similar functionality in the new release, try to use the PeopleSoft objects and drop the customization.

Good communication may be needed to explain that, by adapting PeopleSoft's solution, we bring our system closer to "Vanilla." In cases like these, it is also important to ensure that the upgrade team is familiar with all customizations made to the application. Good documentation certainly helps. It may also be a good idea to involve the people who made the customizations in the upgrade project.

KEY POINTS

1. Do not forget to document all changes.
2. When modifying a subpanel, keep in mind that this modification will affect any panel to which the customized subpanel belongs.
3. In the global development environment use `RFT Short` or `Long` descriptions when assigning a label to a panel field.
4. It is always important to test all modifications.
5. Even a simple addition to the delivered panel will impact future application release upgrades.

CHAPTER 22

Adding new fields and panels

In the previous chapter, we discussed how to add a field to a panel when the field already exists in the PeopleSoft-delivered application and belongs to the record linked to this panel. We also discussed possible implications on future upgrades.

What if the new field does not exist in your current system? When adding such a field to a delivered panel group, you have to make an important decision about the best way to perform your customizations. We have already discussed the advantages of the "Add vs. Modify" approach. Let's now consider practical examples and talk about our customizations in details.

Let's turn to exercise 2:

> Add three custom fields to the employee's Job Data and Job Data Hire panel groups.

Let's say we have three new fields: the Acquisition Date, the Union Seniority Sequence, and the Badge ID. The fields will be populated by data entry via an effective-dated on-line panel.

Let's assume, too, that the team involved in a Fit/Gap analysis has already recognized the fact that the required fields were not delivered by PeopleSoft.

In order to perform this customization we first must identify objects that have to be customized or created. We already know that our task will include creating three custom fields. Our next step is to figure out where to place these fields and what alternatives we may have in implementing this task.

22.1 WHAT OBJECTS SHOULD BE CUSTOMIZED OR ADDED?

From the user requirements, we know that the new fields should belong to the Job Data panel group. Let's open this panel group from the Application Designer and inspect the records linked to the panels in this panel group.

Navigation: GO →PeopleTools →Application Designer →Open →Panel Group → JOB_DATA

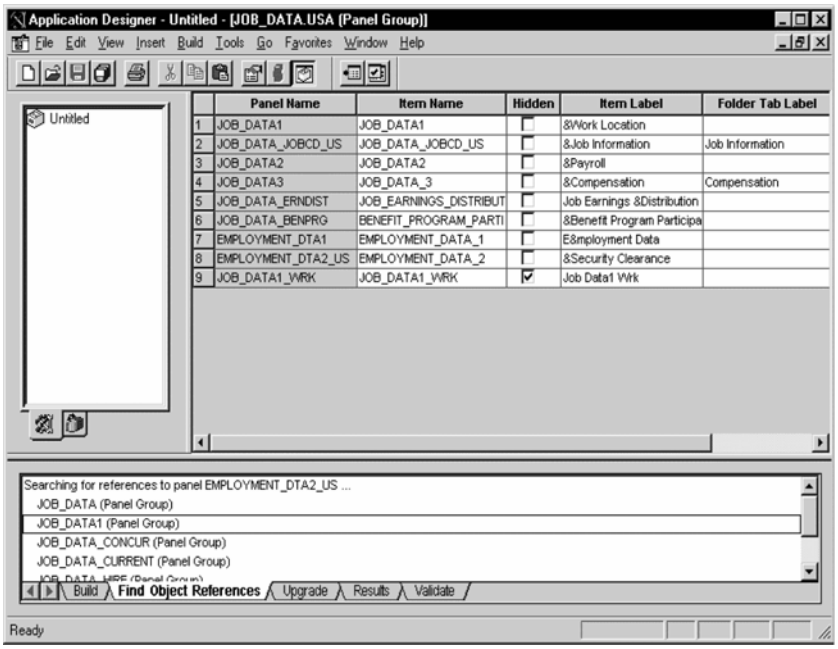


Figure 22.1 Inspecting the JOB_DATA panel group

As you can see from figure 22.1, the JOB_DATA panel group consists of nine panels. This panel group contains employee job history, employment, payroll, compensation, and other information. If you open these panels one by one, you see that the first six panels in this panel group are effective-dated (they include EFFDT as a high-order key field), which supports the maintenance of the employee's job history. Since our new fields should belong to effective-dated panels, we have the following alternatives:

- add the new fields to an existing effective-dated record and the corresponding panel
- create a custom effective-dated record and add this record to one of the existing effective-dated panels in the JOB_DATA panel group
- create a custom effective-dated record and a new panel, and add the new panel to the JOB_DATA panel group

Let's discuss each option:

The first option is probably best from the user's point of view, but as we already discussed in chapter 20, we should stay away from customizing major PeopleSoft-delivered records.

The second option is as good as the first one from the user's point of view if we can find a panel that is not overly crowded and is logically suitable to hold the new fields. There is, however, one important drawback. We know that our fields should belong to an effective-dated panel. We also know that PeopleSoft does not allow multiple records (besides derived/work and related display) within the same scroll bar on the panel. (See part 2 of this book for more details about designing panels with multiple scroll bars.) Since all effective-dated panels have scroll bars, the task becomes a bit more complex. We can possibly add another scroll bar to one of the effective-dated panels, but this involves extensive panel modifications.

Considering the impact on future upgrades, it is simpler and cleaner just to create and maintain a custom panel that houses all current and future custom fields that require effective date processing.

Therefore, we select the third option of creating a new effective-dated record and a panel as our customization approach. The advantage is less customizations to delivered objects; the drawbacks are additional objects to maintain and one more panel with which users may work.

Let's find out what records are used in the JOB_DATA1 panel. Double-click on the JOB_DATA1 panel from the screen in figure 22.1 and click on the  button or select Layout →Order.

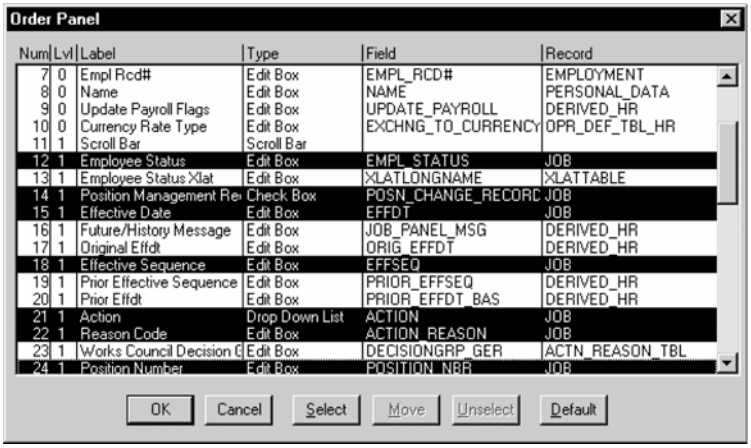


Figure 22.2 Identifying records used in the JOB_DATA1 panel

Since the JOB record is an effective-dated record, it can be used to create our new record definition. We don't necessarily need to find a record to clone. It's just sometimes easier and faster, especially when you know that your new record will have the same key structure as the existing one. We can use a similar approach when cloning effective-dated panels.

To summarize, we identified five new objects: three new custom fields, one custom effective dated record to house these fields, and a custom panel.

After creating a new panel, we need to modify all the related panel groups used to access a new panel. Also, since the new panel is added to these panel groups, the appropriate security access has to be granted to users of the new panel.

22.2 CREATING NEW CUSTOM FIELDS

Let's create our new objects in the Application Designer. We start with the Acquisition Date field (figure 22.3).

After selecting `Date` as a field type for our new field, we are ready to enter other field characteristics as shown in figure 22.4, then save the newly created field as "MY_ACQUISITN_DT."

When a new field is added, don't forget to insert the field into a project by using F7 key or Insert →Current Object into Project. You can also turn on the option to automatically insert a new or modified object to your project as shown in figure 22.5.

Since we want to keep track of all the customizations, we leave this option turned on.

Navigation: Go →PeopleTools →Application Designer →New →Field

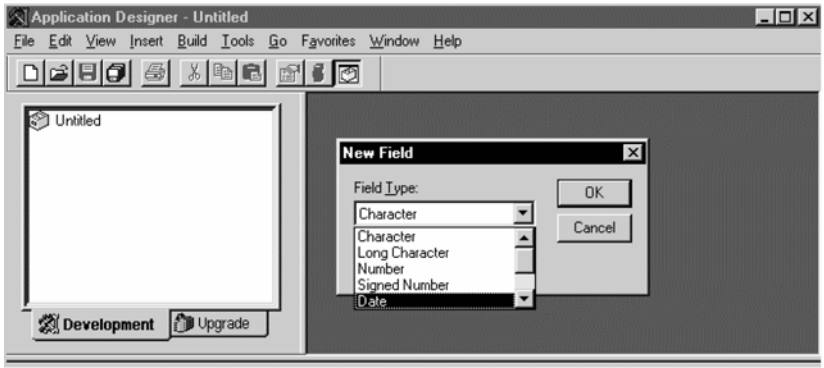


Figure 22.3 Creating a new Date-type field

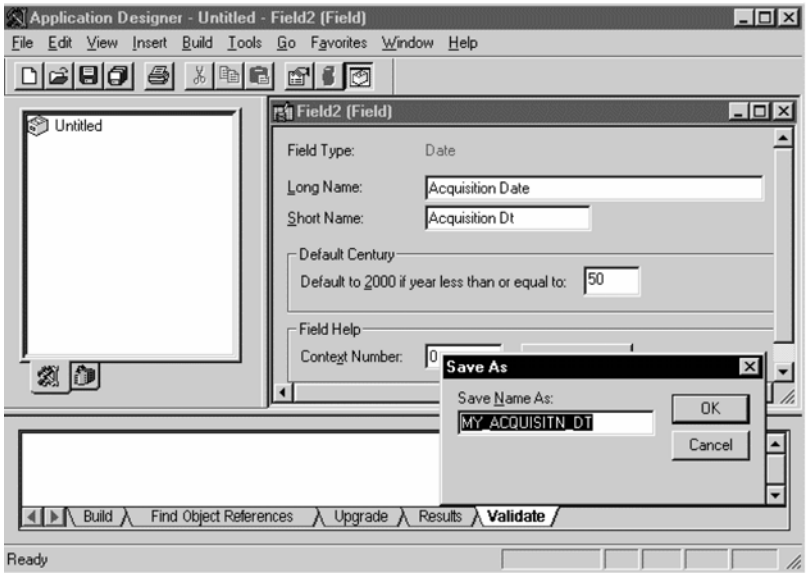


Figure 22.4 Adding new field properties and saving the field as MY_ACQUISITN_DT

Also, since this is the first object in our project, let's save our new project as MY_CUSTOM_02 by selecting File →Save Project As (figure 22.6).

Navigation: Tools →Options

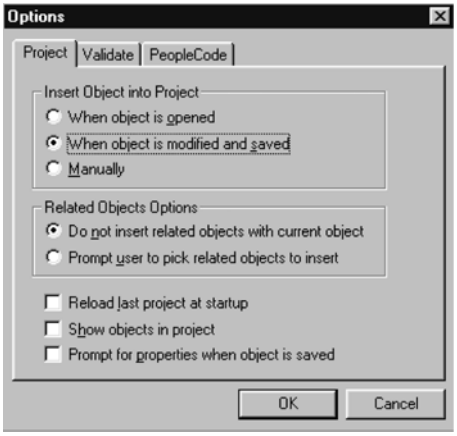


Figure 22.5
This option allows you to automatically insert into the current project any object you save

BP BEST PRACTICE

When working with a single project, use the Automatic Insert option to ensure all customized objects are added to your project.

Do not use the Automatic Insert option if you are simultaneously working on objects that should be placed into different projects. The Automatic Insert option will insert your object into the project currently open.

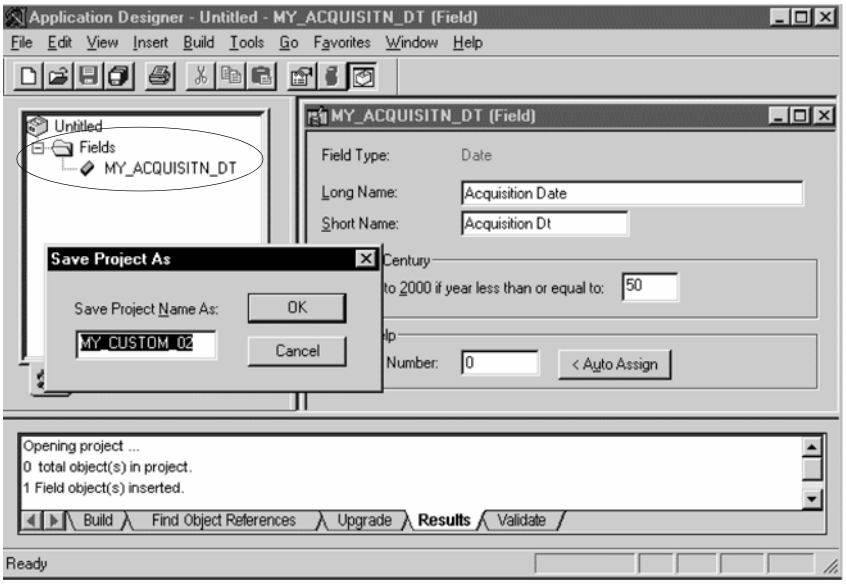


Figure 22.6 Adding a new field to a project and saving the project as MY_CUSTOM_02

Figure 22.7 All the new fields are in the project

Our next task is to create the other two fields by performing the same steps used to create the first field. This time, we create the Union Seniority Sequence field as `Number`, and the Badge ID as `Character`. After creating the new objects, we place them into our MY_CUSTOM_02 project to keep track of all the customizations performed. The panel on figure 22.7 shows our project after we complete these steps.

We just created three new fields and added them to a project. Was it absolutely necessary to create all these new fields? The answer is "No." We could have found some PeopleSoft-delivered fields with the same data types and reused them for our needs. For example, instead of creating the Acquisition Date field, we could have used the FROM_DATE field, and, instead of creating a new Union Seniority Sequence field, we could have reused the SEQ_NBR field.

Both techniques present advantages and disadvantages. If you create custom fields and prefix their names with some specific letters, they can be easily identified as new objects. Also, you can give them meaningful descriptions that can be used in the panels where these fields are placed. This is especially important when dealing with multilingual environments. On the other hand, it's a good idea, if you can, to simply reuse the delivered fields with their properties, since it decreases the number of customized objects, thus saving upgrade efforts. In our particular case, since we wanted to use distinguished labels on the panels, and because this panel will be used with other languages, we purposely created three new custom fields.

22.3 CREATING A CUSTOM RECORD

Creating a custom record is a simple task. Since we've already done all the groundwork and decided to use the JOB record as a candidate for cloning, let's just open the JOB record and save it as MY_JOB_INFO.

Navigation: GO →PeopleTools →Application Designer →Open →Record

Type JOB and press ENTER. Select the Job record and save it immediately as MY_JOB_INFO. After the record is saved, let's leave only the key fields in the record and delete all the fields we are not planning to use. Now is the time to add the three new custom fields to the MY_JOB_INFO Record definition and save it again. It will also be automatically added to the MY_CUSTOM_02 project.

TIP Fields can be deleted from a record definition by highlighting the field and choosing Edit →Delete or pressing the delete key.

The new record definition is created (figure 22.8).

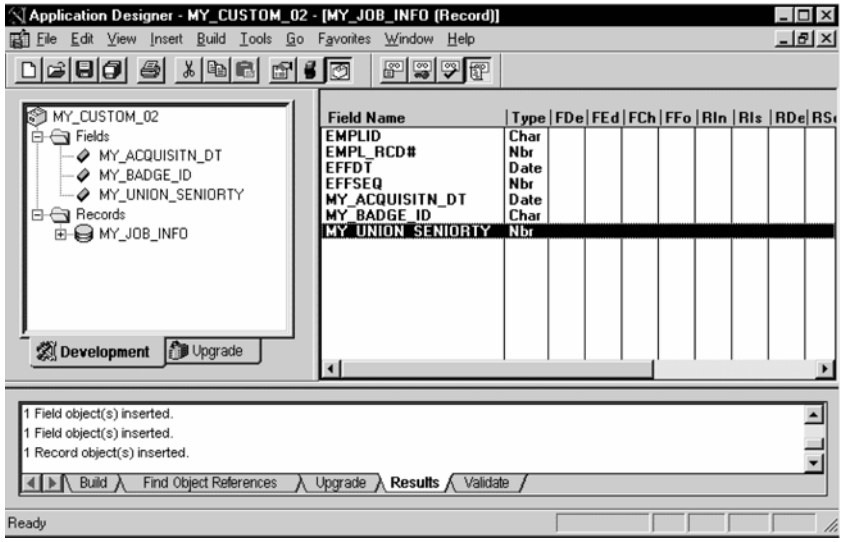


Figure 22.8 Creating a new record definition by cloning the Job record and adding custom fields

As we discussed in part 2, after a record definition is created, the next step is to build a corresponding table in the database. Select Build →Current Object from the Application Designer Menu. Specify Create Tables and Execute SQL now as shown in figure 22.9.

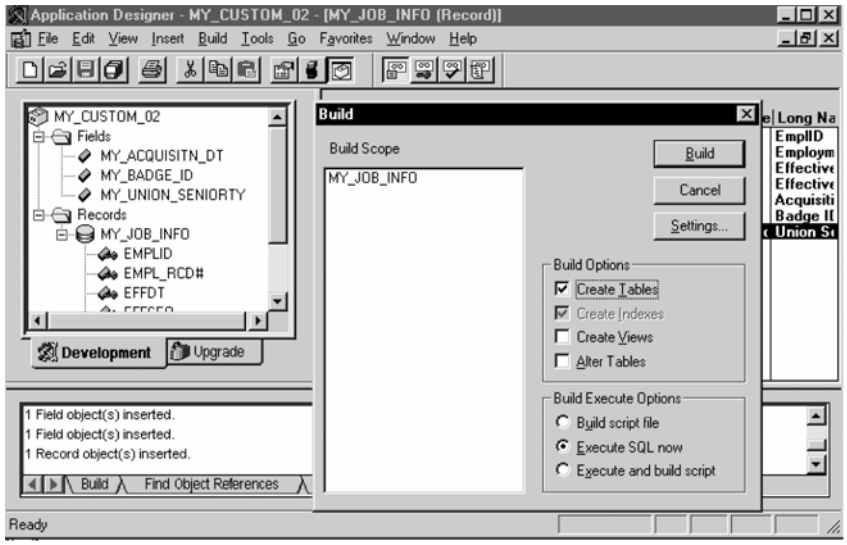


Figure 22.9 Creating a database level table

Since we are creating a brand-new table, it is safe to use the Create Tables and Execute SQL now options.

WARNING The Create Tables option will result in the loss of data in the table if this table already existed.

Since we just created this table, we should not worry about the data. Therefore, after clicking on the Build button, our new table MY_JOB_INFO is created. Please note that you may not have the authority to execute an SQL directly. In this case, you can use the option of building a script file, and your database administrator will execute it for you.

22.4 CREATING A CUSTOM PANEL

We've already decided to create a custom panel named MY_JOB. Using our preferred technique of cloning PeopleSoft-delivered objects, let's find an appropriate panel to clone. It sometimes may take you more time in searching for an appropriate object to clone than to create an object from scratch. Each particular case, creating from scratch or cloning, has its pros and cons. When adding a new panel to a panel group, there is an advantage to cloning an existing panel. First of all, you already know all panels in the panel group. Secondly, your new panel must have the same Level 0 record as the other panels in the panel group to be able to use the same search record specified for all panels in the panel group. Since we've decided to make our new panel effective-dated, our record should also have the same Level 1 record as other effective-dated records in the panel group.

Therefore, in our particular case, we'll clone one of the panels from the JOB_DATA panel group. Let's open, for example, an effective-dated panel named JOB_DATA_JOBCD_US and save it as MY_JOB panel.

WARNING You must never open a PeopleSoft object, make changes to it, and then save under a new name. You must open it, save it under a new name, make changes, then save it again. Doing this any other way is just far too dangreour. By making changes first and then saving the panel, you may accidentally overwrite the delivered panel used for cloning.

We leave all Level 0 fields plus all Level 1 fields in the panel. In our particular case, the EMPLID and EMPL_RCD# are Level 0 fields, while the EFFDT and EFFSEQ are Level 1 fields. All other fields should be deleted.

Let's add our three new fields—the Acquisition Date, the Union Seniority Sequence, and the Badge ID—to the panel (figure 22.10).

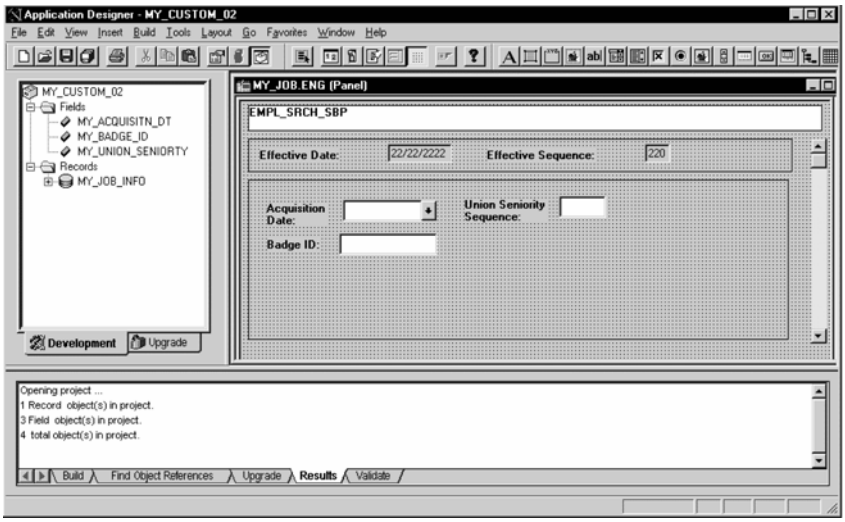


Figure 22.10 Adding custom fields to the new MY_JOB panel

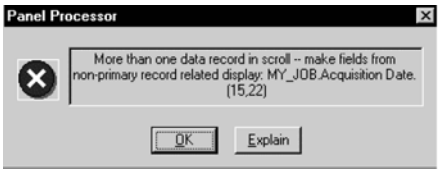


Figure 22.11 Trying to save the newly created custom panel

Once we have all the required fields in our new panel, let's save the panel. After clicking on the button, or selecting File →Save, the Panel Processor displays the error message shown in figure 22.11.

Don't panic! Let's examine the panel layout and fix the problem.

BP BEST PRACTICE

After a panel is created or modified, it is always a good habit to verify the Order panel by clicking on the button or by selecting Layout →Order from the Application Designer Menu.

The Order panel shows all the fields and records as well as the levels to which these fields belong. It also displays the tab order (figure 22.12).

Now it's clear what the Panel Processor did not like about our panel. The JOB and the MY_JOB_INFO records are both located under the same scroll bar. How are we going to correct this? We know that our new fields belong to the MY_JOB_INFO record. This record also has EFFDT and EFFSEQ as its key fields. Should we just change the record behind the EFFDT and EFFSEQ fields from the JOB record to MY_JOB_INFO record?

Don't rush to a solution. In fact, this is a common problem for new PeopleSoft developers. Let's discuss this situation. Suppose we do this change, and now all the

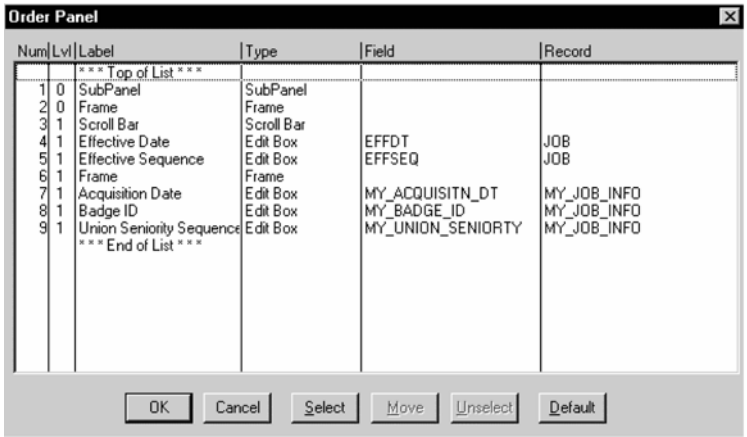


Figure 22.12 The Order Panel for our newly created MY_JOB panel

records in the scroll Level 1 belong to MY_JOB_INFO. The Panel Processor sees no problems and allows you to save this panel. But when you add this panel to the JOB_DATA panel group and start testing, you immediately spot that this panel does not work as it should. It acts as a stand-alone panel, while our users wanted it to be a part of the Job history. Take a look at the panels in the JOB_DATA panel group. All effective-dated panels have the same dates. Why? Because all Level 1 information comes from the same JOB record. If we make our Level 1 record different from the JOB record, our panel will maintain its own dates, one that would have nothing to do with the effective date of the JOB record.

Therefore, the correct solution is to add one more scroll bar and make our new fields belong to the Scroll Level 2. Let's do it. Select Insert →Scroll Bar from the Application Designer menu and place it next to our new fields as shown in figure 22.13.

After the new scroll bar is inserted, we need to set its Occurs level to 2. Let's click on the new scroll bar and, with a right mouse click, select Panel Field Properties. Specify the label of the new scroll bar as Scroll bar MY_JOB (figure 22.14).

The next step is to switch to the Use tab of this panel and specify the MY_JOB scroll bar properties (figure 22.15).

Since we created the scroll bar for the sole purpose of maintaining another record under a different level, we don't really need to show this scroll bar to our users. Therefore, we'll make it invisible. Also, it is a good idea to select the No Row Insert and No Row Delete options. While this functionality is not needed for our panel, if not turned off, it may still be used by mistake. After saving the Panel Field Properties information, we have to make sure that all our new fields belong to this scroll bar level.

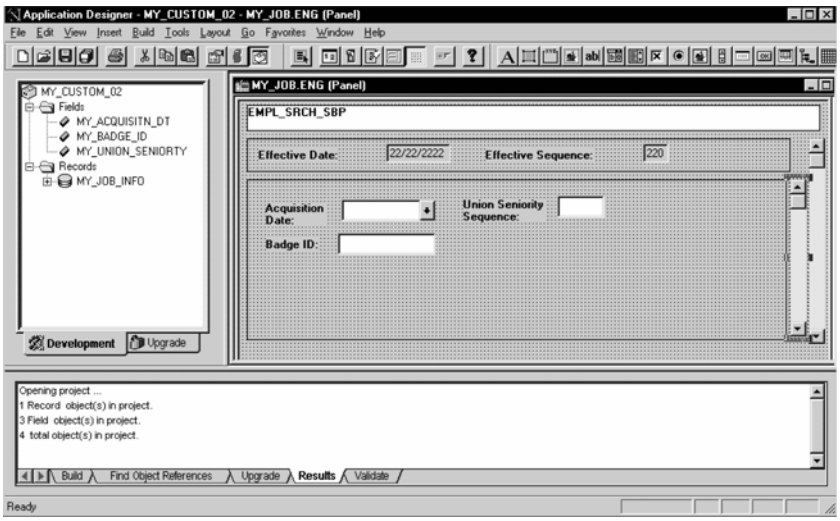


Figure 22.13 Adding a Level 2 scroll bar to a panel

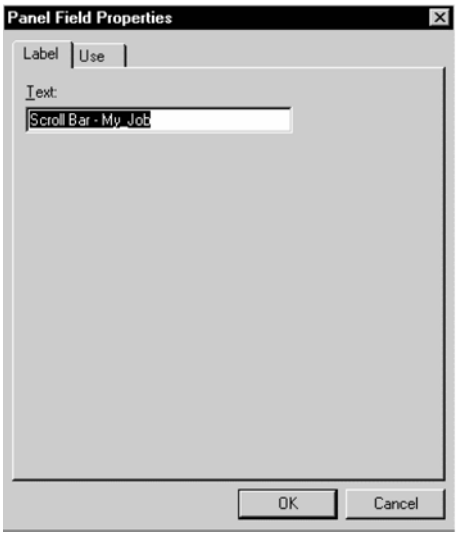


Figure 22.14 Specifying a label for a new scroll bar

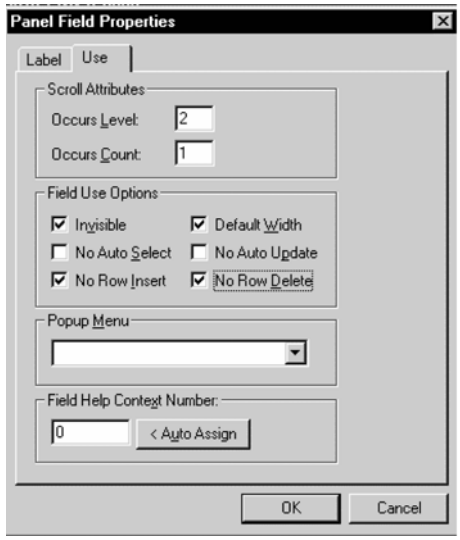


Figure 22.15 Specifying the Occurs Level 2 for our new scroll bar

---

TIP Placing a scroll bar right above the group of fields makes all the fields that follow the scroll bar belong to the same scroll level.

Figure 22.16 shows that fields number 8, 9, and 10 as well as the scroll bar itself, now belong to Level 2.

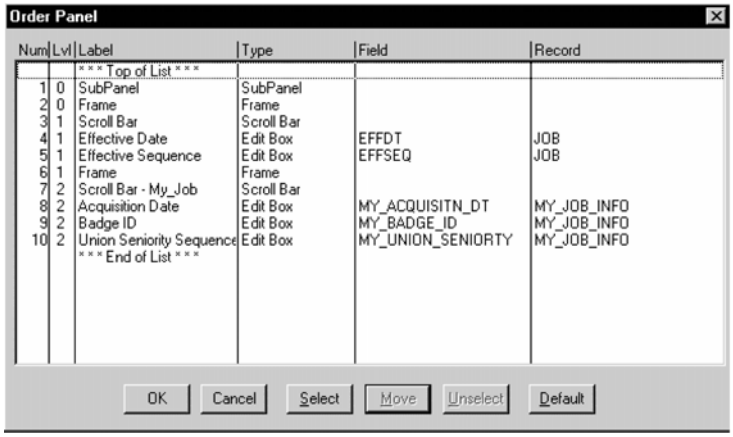


Figure 22.16 The scroll bar My_Job is located right above all the custom fields

If you look at figure 22.16, you'll notice that the Badge ID field is located above the Union Seniority Sequence field, while on the panel the order of these fields is opposite. To make the order of tabulation the same as the field order on the panel, highlight the Badge ID field and click on the Select button. The field disappears from the screen. Don't be alarmed. Just highlight the Union Seniority Sequence field and click on the Move button. Our Badge ID field will be moved as shown in figure 22.17.

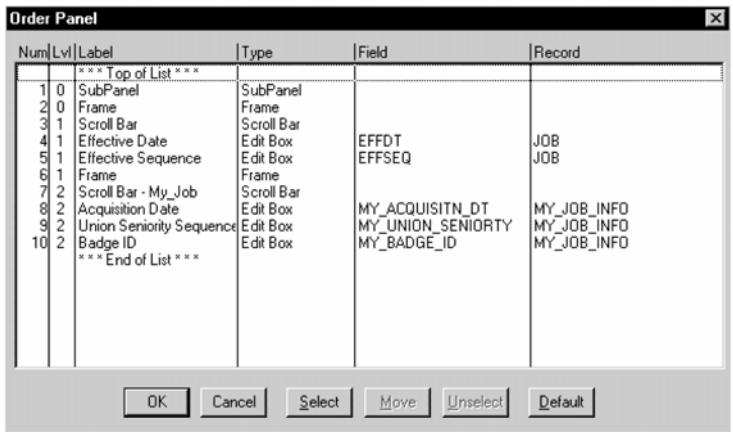


Figure 22.17 Changing the field order in the Order Panel

After all is done, let's save our new panel; it will be added to the MY_CUSTOM_02 project.

Now it is time to test the panel. Just click on the button or select Layout → Test Mode, and you can see how the panel looks. You can also perform a preliminary tab order test by entering values and using the tab key to switch between the fields (figure 22.18).

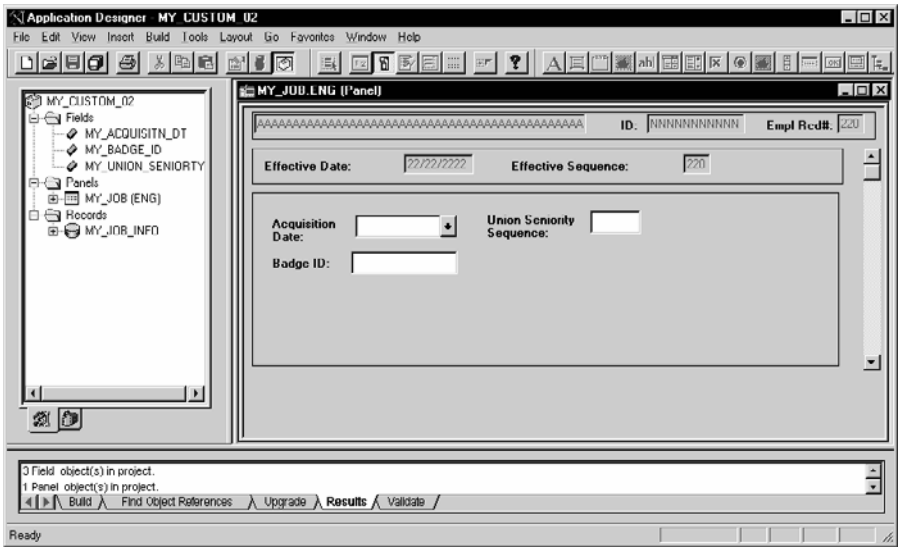


Figure 22.18 Using Test Mode to test the new panel

In order to perform a real test of the panel, we would need to finish all other related modifications.

22.5 ADDING A NEW PANEL TO THE EXISTING PANEL GROUP

Based on the original request, the new panel should be accessed from both the JOB_DATA panel group and the JOB_DATA_HIRE panel group.

First, let's open the JOB_DATA panel group.

We add our new panel at the end of all effective-dated panels, right above EMPLOYMENT_DTA1. In order to add our panel, highlight the EMPLOYMENT_DTA1 panel, and select Insert →Panel Into Group from the Application Designer menu. Type MY_JOB panel name, highlight the panel, click on Insert, and then click on Close. Our new panel is inserted into the JOB_DATA panel group (figure 22.20).

Navigation: Go →Application Designer →Open →Panel Group →JOB_DATA

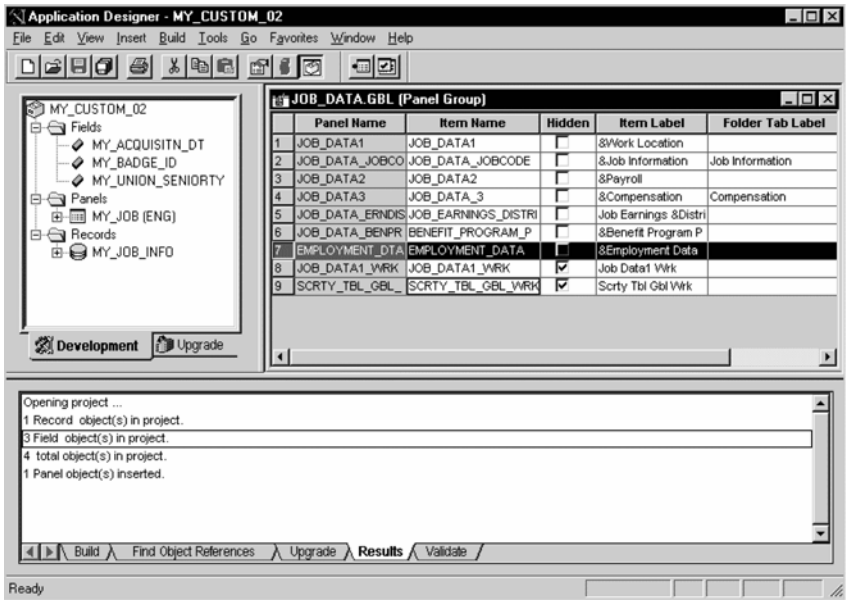


Figure 22.19 The JOB_DATA panel group

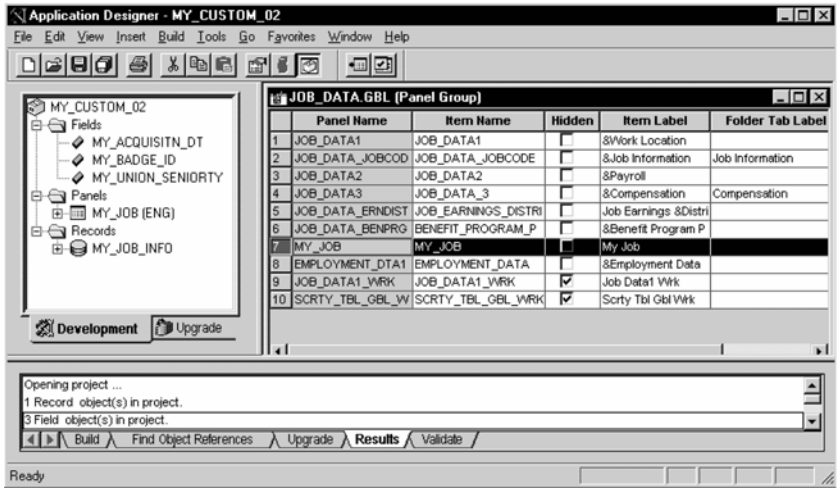


Figure 22.20 Inserting the MY_JOB panel into the JOB_DATA panel group

After the panel is inserted, we save the modified panel group, and it is inserted into our project.

Our next step is to add the panel to the JOB_DATA_HIRE panel group. We can demonstrate another method of adding a panel to a panel group by dragging the panel from the project workspace. Let's open the JOB_DATA_HIRE panel group, select the MY_JOB panel from the project workspace, and drag it to the panel group (figure 22.21).

TIP The drag-and-drop method inserts your panel (from the left side on your screen) right before the highlighted panel (on the right side of the screen). If no panels are highlighted, the panel is added to the end.

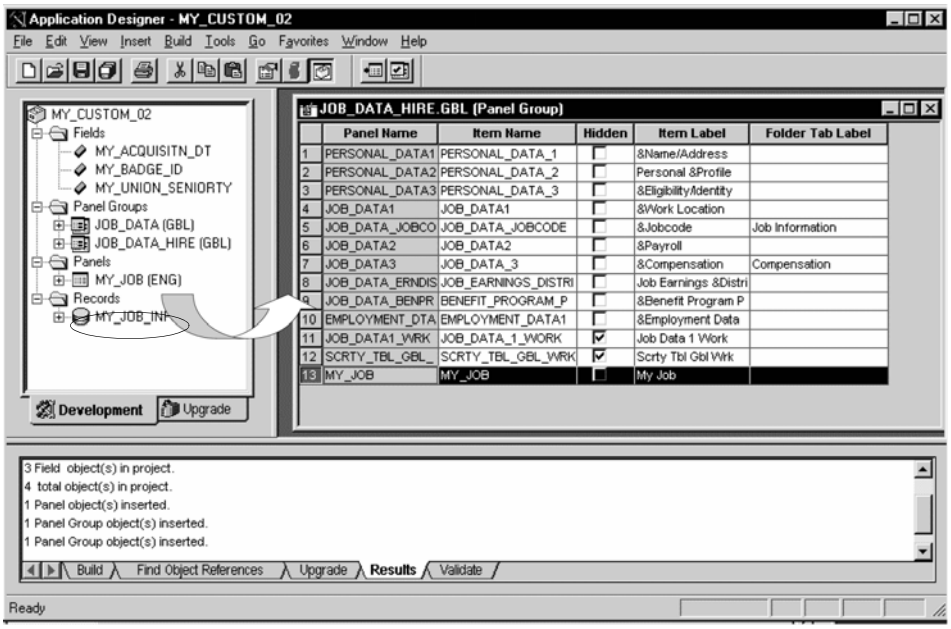


Figure 22.21 Using the drag-and-drop method to insert a panel to a panel group

Since we have not specified the exact place in the JOB_DATA_HIRE panel group where we wanted the MY_JOB panel to be inserted, the system simply placed our panel at the end as number 13 (figure 22.21). You can change the order of panels in the panel group by dragging the panel to the place you need it to be.

When all is done, let's save our modified panel group. Our project will look as shown in figure 22.22.

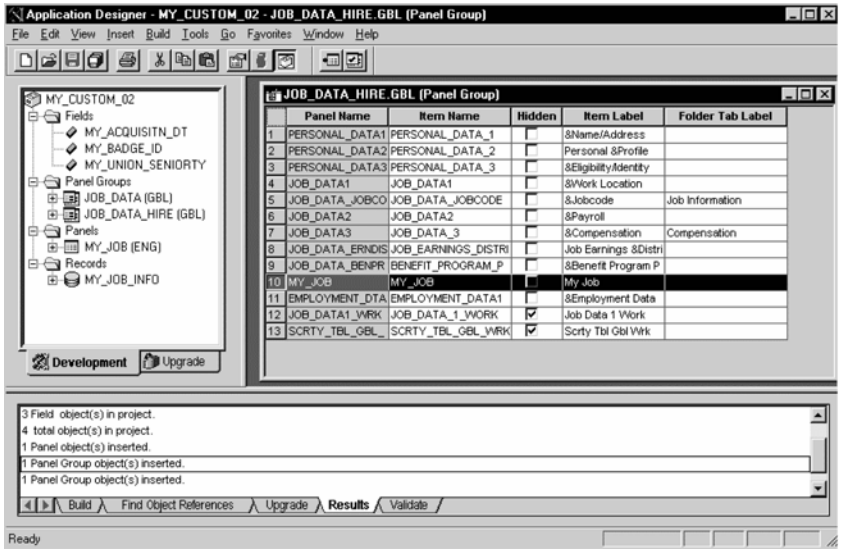


Figure 22.22 Our project contains two modified panel groups

22.6 GRANTING SECURITY ACCESS

Since we changed the existing panel group by adding a new panel to it, we have to grant security access to all users who would need to work with this new panel. First, we grant security access to the ALLPANLS operator class that will be used to test our customizations.

Select Menu Items, and double-click on the ADMINISTER_WORKFORCE_(GBL) menu (figure 22.23).

Navigation: Go →PeopleTools →Security Administration →Open →ALLPANLS

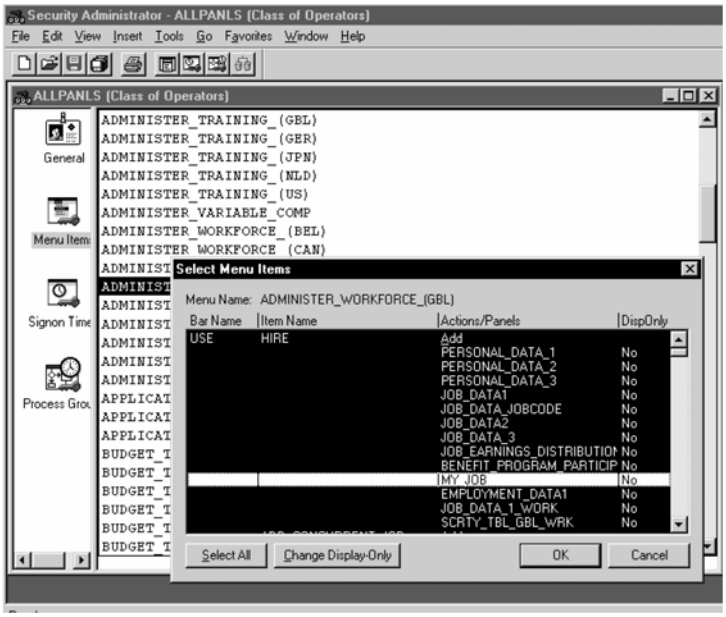


Figure 22.23 Selecting ADMINISTER_WORKFORCE_(GBL) menu item

Highlight the MY_JOB panel group under the bar name USE and item name HIRE. Repeat the same process with the JOB DATA item name, press OK and then, save the modified security. Now security access has been granted to all users who need to use this panel.

22.7 TESTING OUR CHANGES

We need to make certain that all our modifications work correctly. Remember, we created a new panel and a new record. We also added a new panel to the existing panel group. Our goal is not only to verify that the new functionality is in place, but also to ensure that it works perfectly with the delivered objects.

Let's start by testing the Job Data panel group.

Navigation: Go →Administer Workforce →Administer Workforce (GBL) →Use →Job Data →My Job →Update/Display All

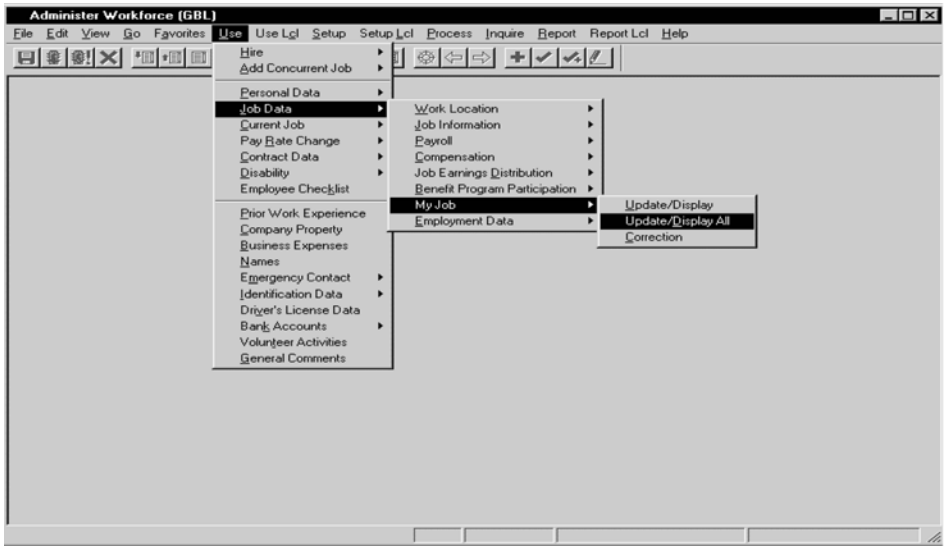


Figure 22.24 Testing the Job Data panel group

Let's select an employee—for example, "Smith, Lily"—and fill in our new panel as shown in figure 22.25.

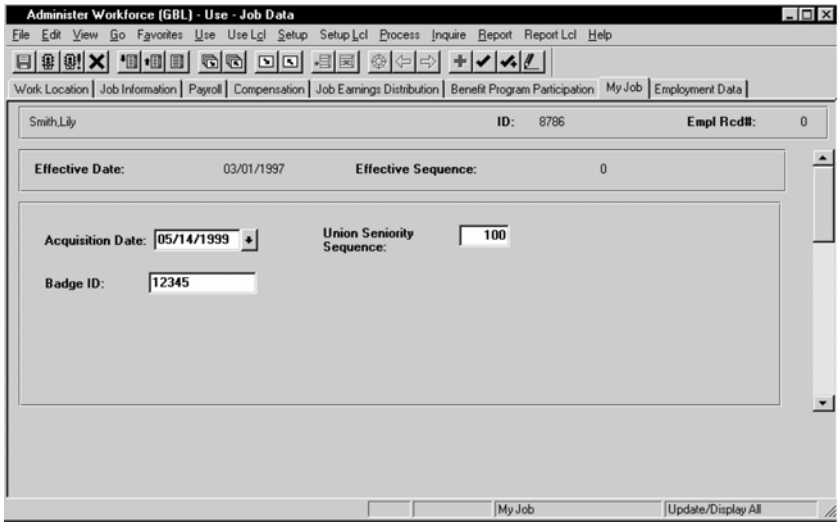


Figure 22.25 Entering information into the new panel

Pay attention to the Effective Date and Effective Sequence fields. These fields have exactly the same data as that in the JOB record. If we insert another JOB record with another effective date, the data in our panel should also change. Let's test this. Figure 22.26 shows that when a new JOB record is inserted, the effective date on MY_JOB panel is also changed while the rest of the data are carried over from the previous record.

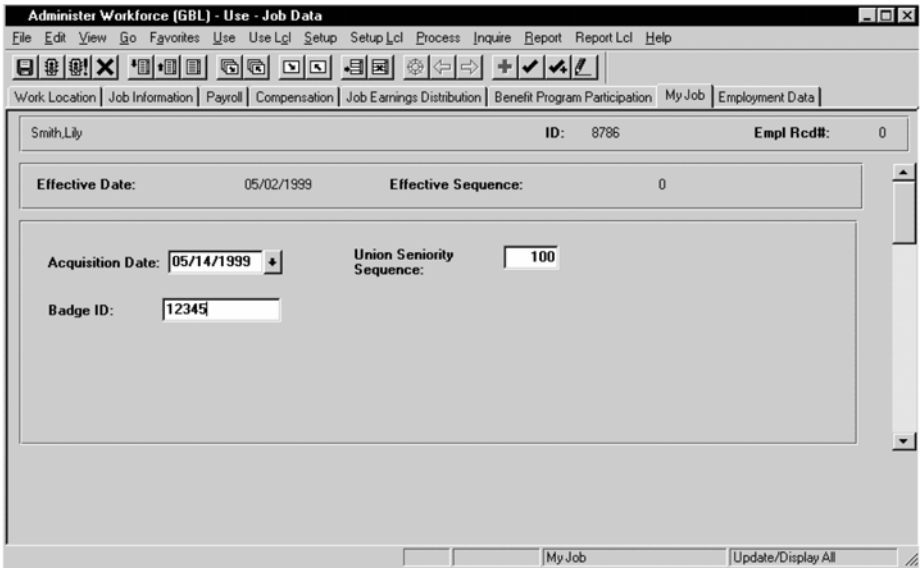


Figure 22.26 Inserting a JOB row leads to the insertion of another row in the MY_JOB panel as well

Since we know that we created a new custom record MY_JOB_INFO to save information on the panel, let's verify that the information is saved correctly and that it is also effective-dated. In order to do this, we just execute a simple `Select` statement (figure 22.27).

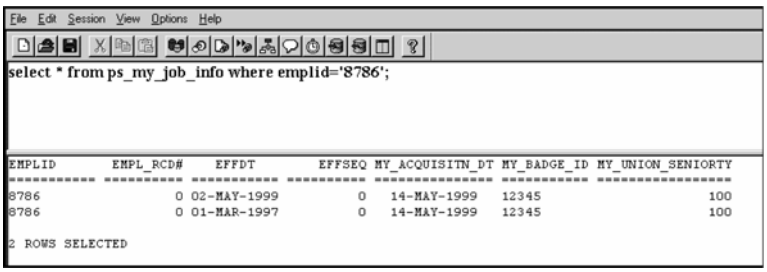


Figure 22.27 Verifying that the panel information was saved correctly in the underlying MY_JOB_INFO table

Use your SQL tools to verify that the underlying table is properly updated by any online panel operations.

In order to verify all possible situations, a complete test plan would have to be created and executed. For example, we would need to test our customizations in all available modes: `Update`, `Update/Display`, and `Corrections`. The new fields' boundaries would have to be verified as well.

To test the Job Data Hire panel group, we would follow the same testing sequence that we used for the Job Data panel group.

22.8 POSSIBLE IMPACT ON FUTURE UPGRADES

As our readers may have already noticed, we tried to minimize the impact on future upgrades during all stages of our development. We used distinctive names for all of our objects, documented our changes, and created a project to keep track of all customizations. We also avoided the temptation to customize the delivered JOB record. However, some of the changes to the delivered system were not avoidable. For example, we modified the delivered Panel Groups: JOB_DATA and JOB_DATA_HIRE. Therefore, with the new release, we will either have to re-apply our changes if PeopleSoft delivers any new functionality to these panel groups, or to accept our customized version if PeopleSoft makes no changes to them. Also, as we already emphasized in our previous discussion, there is always a possibility that PeopleSoft may deliver similar or even the same functionality. In this case, PeopleSoft's new features should take precedence over ours.

KEY POINTS

1. When creating a custom record, you can either re-use the existing fields or create custom ones.
2. You can insert an object into a project by using the F7 key or Insert →Current Object into Project. You can also turn on the option in the Application Designer to automatically insert a new or modified object to your project.
3. The Application Processor will not allow you to save a panel with more than one data record in a scroll. Exceptions are fields from Derived records and related display fields.
4. When modifying panels, always examine the Order Panel to ensure that all the fields are located in the correct places and belong to all the proper scroll bars.
5. All panels in the panel group should have the same Level 0 records.
6. When adding an effective-dated panel to a panel group where all panels have the same effective date, make certain that the effective date field belongs to the same record in all the panels.
7. While testing panel modifications, it is essential to verify (with the help of your SQL tools) that the underlying table is properly updated by any online operations.

CHAPTER 23

Adding new functionality to PeopleSoft-delivered applications

In this chapter, we will show you how you can add new functionality to an existing application by cloning PeopleSoft-delivered objects. Some knowledge of the PeopleSoft Benefits Administration module will help our readers understand the business reason for the modification under discussion. For those readers not familiar with Benefits Administration, this exercise may be useful from a purely technical development perspective.

Again, let's turn to an example, exercise #3:

> Allow users to delete the Benefits Administration Event from an online panel based on the user's selection. Only events that are not finalized should be allowed for deletion.

Our task here is to allow users to delete certain rows from the database. Only super-users with special security access will be allowed to perform this function.

Why would we need to develop a process that allows the deletions of the Benefit Administration events? Let's examine the background of this problem.

Benefits Administration is a PeopleSoft application that helps administer employee benefits in an automated fashion. Benefit tables are populated with eligible benefits for employees through a batch process, and employees are then allowed to make their choice. The Benefits Eligibility process makes use of related employee information from the PeopleSoft Human Resources application. In this process, a user can correct or delete information that was incorrectly entered into an HR application. Such actions can result in incorrect benefit eligibility information that was prepared based on the original HR information. PeopleSoft Benefits Administration process disconnects when the HR information is deleted, or does not reprocess the event correctly when key HR information is changed. Therefore, if we allow our Benefit super-users to delete an incorrect event and then reprocess the event, the benefit eligibility information will be corrected.

23.1 WHAT OBJECTS SHOULD BE CUSTOMIZED OR ADDED?

Since this functionality is not currently available within the Benefits Administration module, we will create a custom panel to allow our users to perform the changes online. A custom record should also be created and linked to the new panel. When a user selects a particular event for deletion, SQL delete commands will be executed. We'll place these commands into appropriate record field events, thus creating a custom PeopleCode program.

23.2 CREATING A CUSTOM RECORD BY CLONING AN EXISTING ONE

As usual, it is useful to find an appropriate record from which to clone. Since we are developing this project for Benefits Administration, let's take a look at the main Benefits Administration table, BAS_PARTIC (figure 23.1).

This table contains the information about employee's benefit events. Since our task is to show users all available events for a particular employee and allow them to select from a list of events, we create a view from this table.

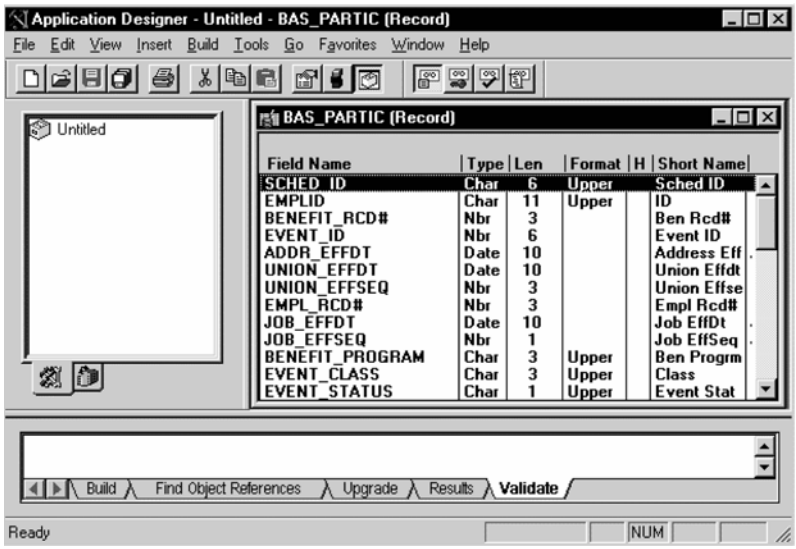


Figure 23.1 Selecting the BAS_PARTIC record as a source for cloning

Let's first save this record as MY_BAS_DEL_VW and then delete all the fields we don't need to use. Our next step is to change the record's property to SQL View. The SQL View `Select` statement looks as shown in figure 23.2.

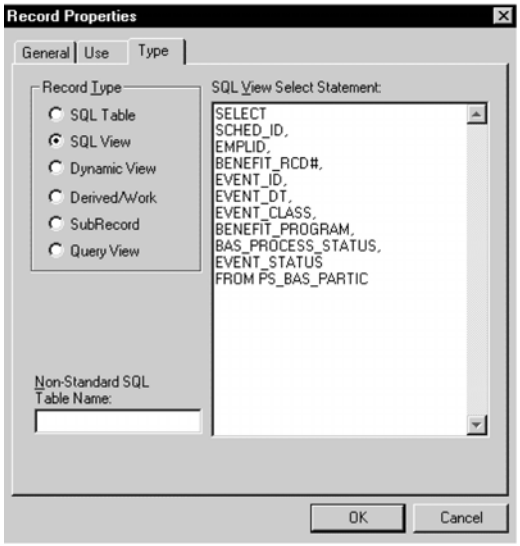


Figure 23.2
Creating a custom view definition

Before saving our new record definition, let's compare all the fields we specified in the view definition to the fields in our record definition. Please remember that the field order in the `Select` statement shown in figure 23.2 should exactly match the order of fields in our new record definition. Let's also not forget to put a useful description for our new view definition in the General tab of Record Properties.

It is important to define the key fields for the view. Here we defined SCHED_ID, EMPLID, BENEFIT_RCD#, and EVENT_ID as key fields.

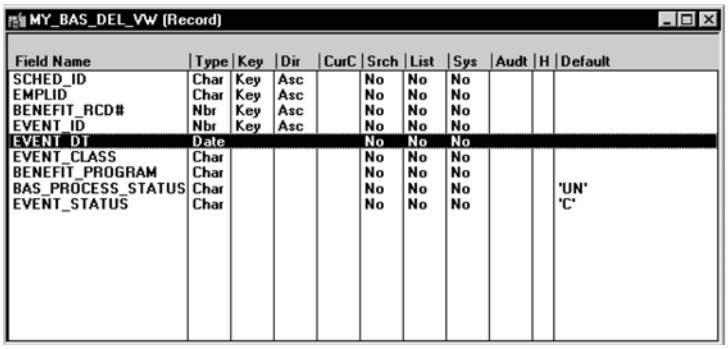


Figure 23.3 Defining the key fields

Let's save our record definition. After the record definition is created, we need to build the actual view in the database. Let's build it by clicking on the tool bar button or selecting Build →Current Object from the Application Designer Menu (figure 23.4).

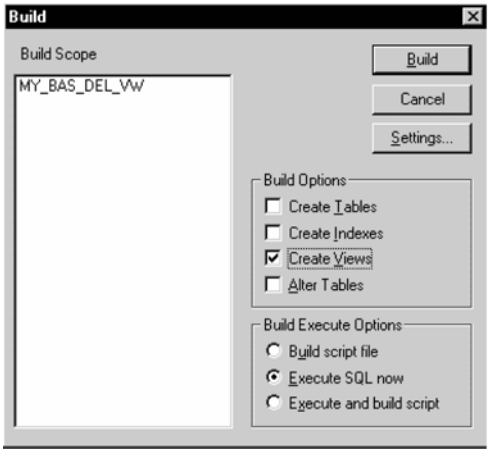


Figure 23.4
Creating a database level View

Select the Create Views checkbox from the Build Options group box. Also click on the Execute SQL now radio button from the Build Execute Options group box. Then, click on the Build button to execute the SQL and create the view. The view is created, and the information about the Build process is displayed on the Application Designer output window, as shown in figure 23.5.

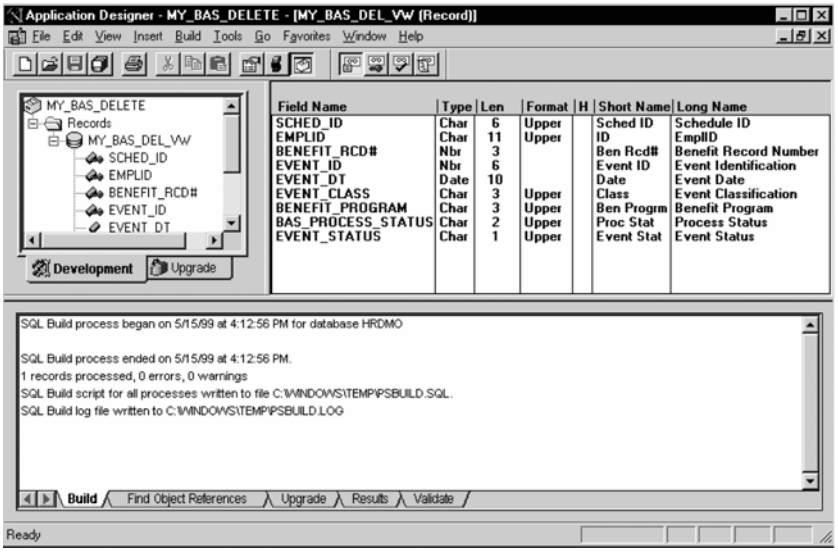


Figure 23.5 The information about the Build process is displayed

---

TIP You can create an SQL View at any point in time without any fear of data loss.

As you can see from figure 23.5, our first object created for exercise 3 is added to a project automatically because we specified an automatic insert to a project in the Tools →Options. We also saved our new project as MY_BAS_DELETE.

Now that the view is created, we can construct a custom panel.

23.3 CREATING A CUSTOM PANEL

Let's look at the PeopleSoft-delivered Benefits Administration application and find a panel that may be used for cloning. Bear in mind that our new panel must be an employee-level panel and should contain the information about various employees' benefit events.

TIP As a basis for cloning, choose an application panel that resembles the functionality required for the customizations.

Let's open the Event Status Update panel to see if this panel could be used as a basis for cloning.

Navigation: Go →Compensate Employees →Administer Automated Benefits →Use →Event Status Update.

Select any employee ID—for example, 8845—and press the ENTER key. The system will display the employee's Event Status Update panel. BAS_PARTIC_STS panel is used as indicated by ❶.

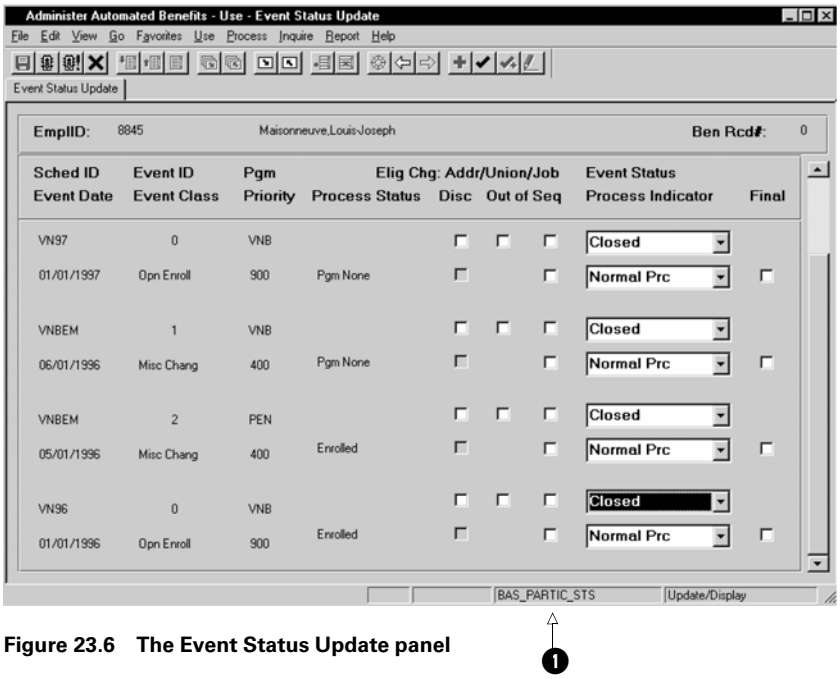


Figure 23.6 The Event Status Update panel

Let's open the BAS_PARTIC_STS panel used in this screen.

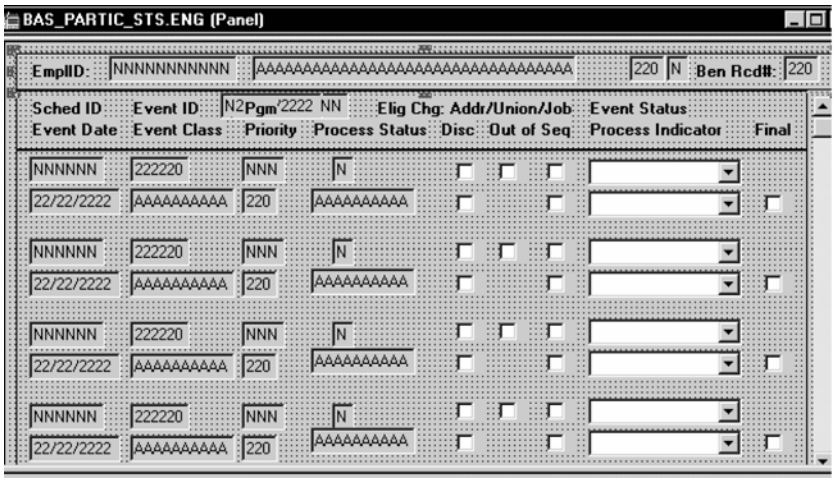


Figure 23.7 The BAS_PARTIC_STS panel

As you can see, the panel in figure 23.7 can be easily used for this task. Let's first save the panel as MY_BAS_DEL_EVNT and then customize it. You have to be really careful when deleting all fields that are not going to be used and replacing the records behind the fields with our custom MY_BAS_DEL_VW record. Sometimes, cloning a panel may become so cumbersome that it may be easier just to create your own panel from scratch. No matter what method you use to create your custom panel—cloning or creating from scratch—your new panel will look like the one shown in figure 23.8.

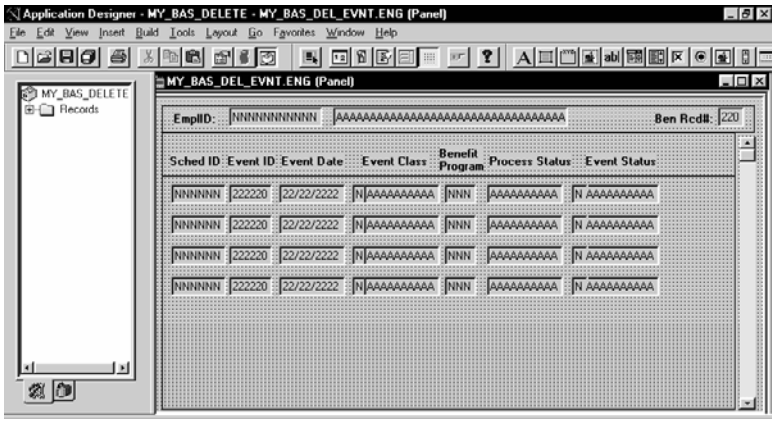


Figure 23.8 Creating a custom panel by cloning the BAS_PARTIC_STS delivered panel

Let's verify that all fields in our panel are in the right order and belong to the correct records. Click on the 12 button or select Layout →Order and examine the field order in our new panel (figure 23.9).

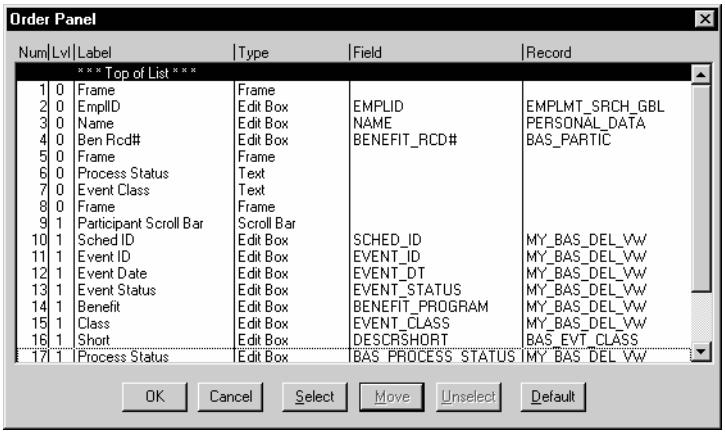


Figure 23.9 The Order panel for MY_BAS_DEL_EVNT

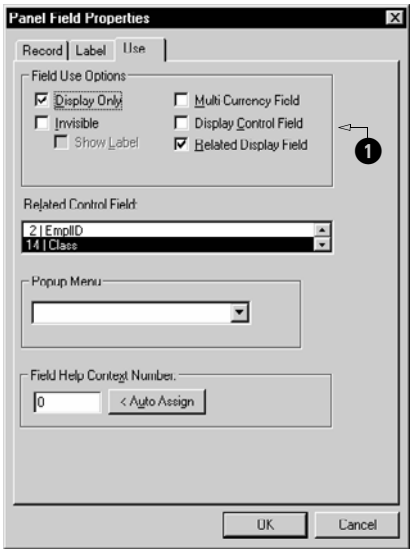


Figure 23.10 Making sure that the DESCRIPTION is a Related Display field

Pay attention to all the levels used in this panel (shown in the second column) and all corresponding records. Our panel consists of two levels. Level 0 houses the EMPLID, NAME, BENEFIT_RCD#, as well as text fields used as headers for the Level 1 columns. Multiple records on Level 0 are allowed since we don't have a scroll bar at this level.

Level 1 requires more attention. We can see two records that belong to this level, MY_BAS_DEL_VW and BAS_EVT_CLASS. Is this a mistake? Take a look at the field column in figure 23.9. The field that belongs to BAS_EVT_CLASS is the DESCRSHORT field. Based on the Display Control and Related Display rule discussed in part 2 of this book, you can include a field from another record under the same scroll bar if the field is specified as a Related Display field. This is, most probably, the case here. Let's verify this by selecting the Event Class description field on the panel and keying CTRL+F or using the right mouse button on the field and selecting

Panel Field Properties (figure 23.10). The DESCRSHORT field is a Related Display field with the Event Class as its Display Control field (❶).

As we expected, the DESCRSHORT field is specified as a Related Display field and has a reference to the EVENT_CLASS field as a Display Control field.

Let's get back to the Order panel in figure 23.9. We also need to verify that all fields listed in this panel are in the correct order. The number in the first column of the Order panel indicates the tab order. Notice that field Event Status is not in the right place. If you use the TAB key, it will position a cursor on the Event Status field right after the Event Date field is tabbed out. According to our panel design, this field should be the last field in a row for the panel. This problem is very simple to fix. Just highlight the field you need to move—in our case, it's field 13—and press the Select button as shown in figure 23.11.

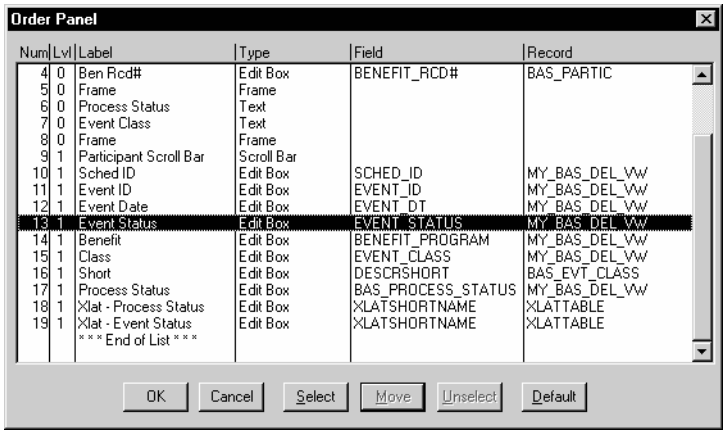


Figure 23.11 Selecting a field to be moved to another place on the panel

Then highlight the field to which you want to move the selected field—in this example, it's field 17—and click on Move. Now the Event Status field is in the right place, located below the Process Status field.

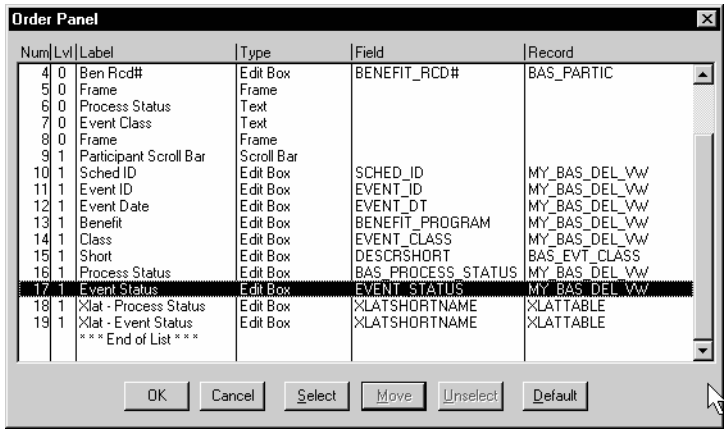


Figure 23.12 Repositioning the Event Status field

Let's take another look at our Order panel. Is everything correct? Looks fine at first glance. Now, let's save our panel changes again (figure 23.13).

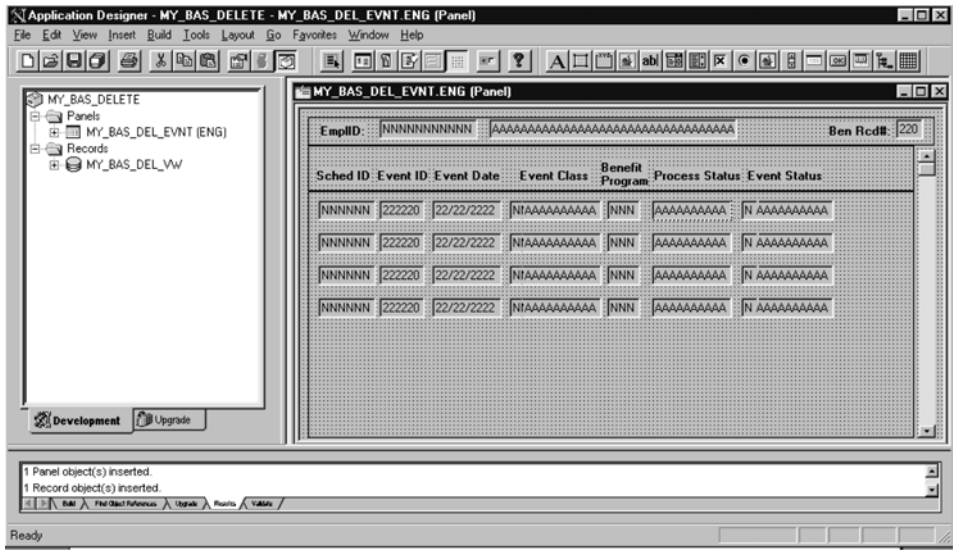


Figure 23.13 The MY_BAS_DEL_EVNT panel is added to our project

Now we can use Test Mode by clicking on the tool bar button or selecting Layout →Test Mode from the menu.

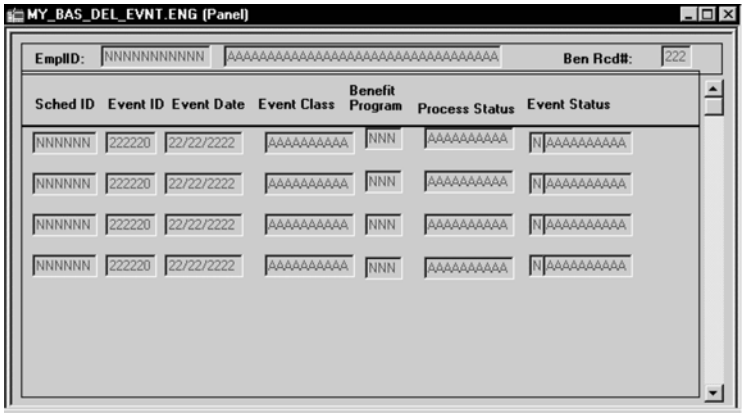


Figure 23.14 Looking at our new panel in test mode

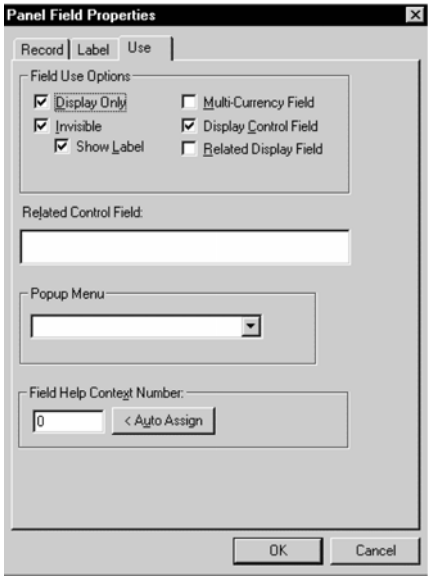


Figure 23.15 Making the Event Status Display Control field invisible

Notice that there are two fields under the Event Status column on the panel: the Event Status Display Control field and the Event Status Description (figure 23.14). We obviously forgot to make our Event Status field invisible. Let's perform this change. First, we have to exit the test mode by clicking on the tool bar button again. Then, right-click on the Event Status field and select the Panel Field Properties. Mark the Event Status Display Control field invisible (figure 23.15). Don't forget, however, to click on the Show Label option; otherwise, it will become invisible as well.

After pressing the OK button, let's test our panel again (figure 23.16).

The Event Status field looks fine now, but how are our users going to delete events? We display all the information about events for users to decide what events should be deleted. Now we need to give them the ability to do so. This is a good time to use a Derived/Work record field as a placeholder for additional functionality.

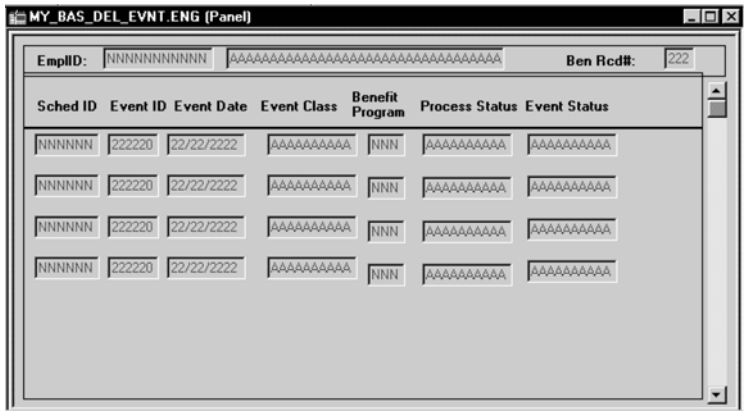


Figure 23.16 The MY_BAS_DEL_EVNT panel in test mode

23.3.1 Creating custom fields for a Derived/Work record

Let's create a custom field MY_EVENT_DELETE as a Yes/No field to be used in the panel for every event selected for deletion.

Navigation: Go →Application Designer →Select New →Field

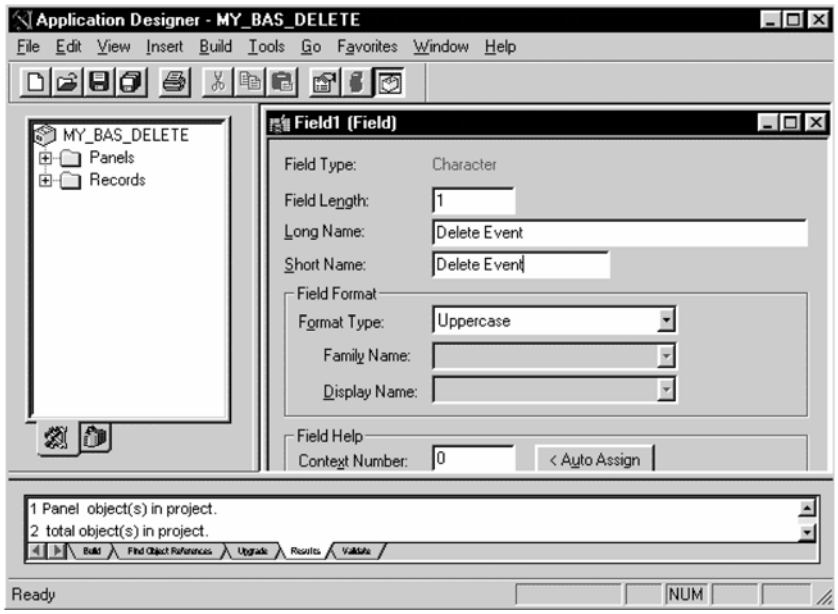


Figure 23.17 Creating a custom field for a Derived/Work record

Let's save the field as MY_EVENT_DELETE. It will be automatically added to our project.

Soon we'll plug special functionality into this field and discuss it in detail, but now, we need to create one more field, MY_DELETE_PROCESS. Why another field? The MY_EVENT_DELETE field will be displayed for each event on the panel. Some events will be marked for deletion, others won't. Users will be able to scroll down the panel and review all the events. When necessary, they will uncheck some events chosen for deletion. After all events on the panel are reviewed, users will need a way to indicate that the deletion process can occur. This is why we need another field.

We repeat the steps described above and create another one character field as a placeholder for our `Delete` script.

Navigation: Go →Application Designer →Select New →Field

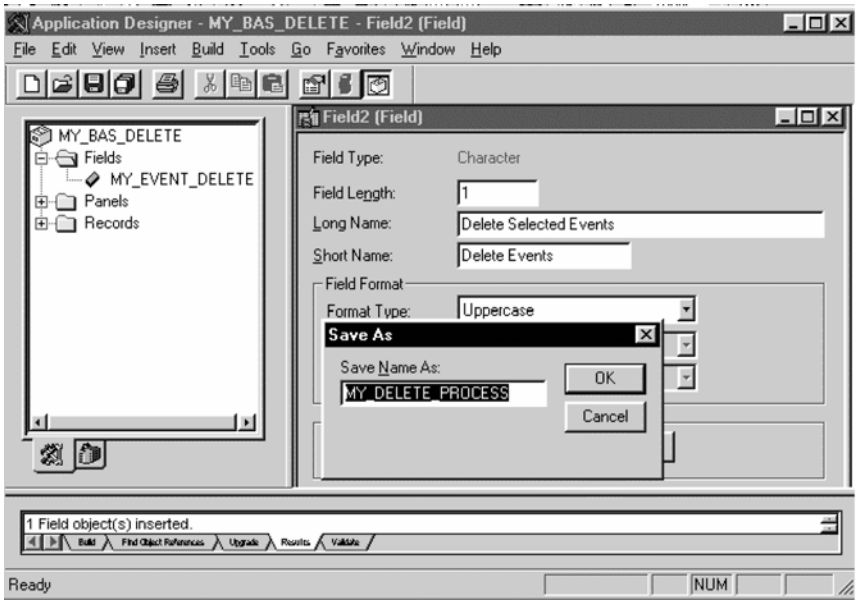


Figure 23.18 Creating the MY_DELETE_PROCESS field

We are now ready to create a custom Derived/Work record to hold our newly created fields.

23.3.2 Creating a custom Derived/Work record

As we discussed in part 2, Derived/Work records are often used to display temporary values on a panel. You can also use them to store temporary values from user input. Since the Application Processor will retrieve only the fields from the Derived/Work

record that are explicitly referenced by the particular panel into a record buffer, you can keep all your work fields for multiple projects in one custom record. Fields in such records are often used to store and trigger PeopleCode programs. Let's create a Derived/Work record MY_WORK.

Insert our two custom fields MY_EVENT_DELETE and MY_DELETE_PROCESS to the record as shown in figure 23.19.

Navigation: Go →Application Designer →New →Record

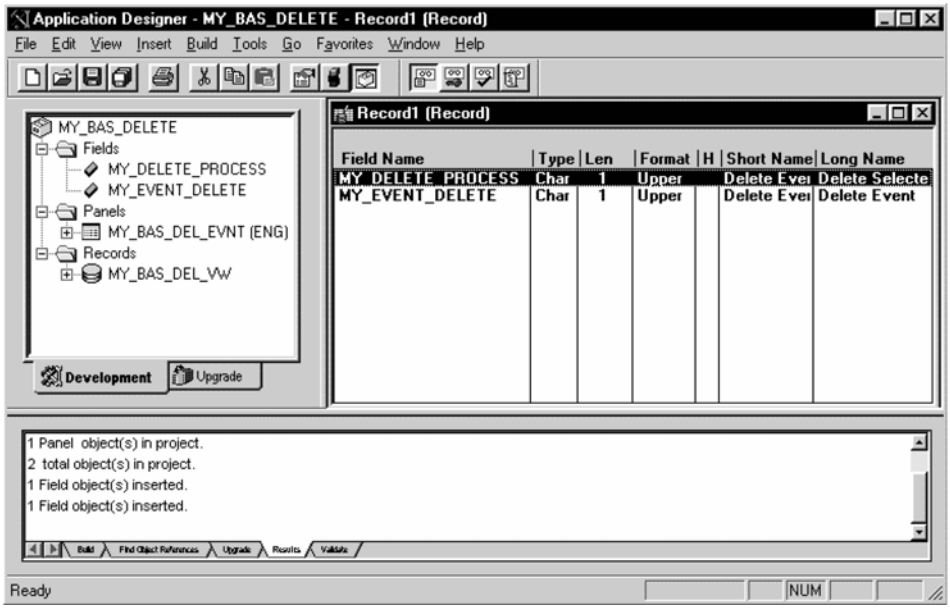


Figure 23.19 Creating a custom Derived/Work record

After the fields are inserted into the record, make certain that the Edit type for MY_EVENT_DELETE field is specified as a Yes/No table edit. Let's save the new record as MY_WORK. Now we need to define the record properties. We have to specify the type record as Derived/Work record (figure 23.20).

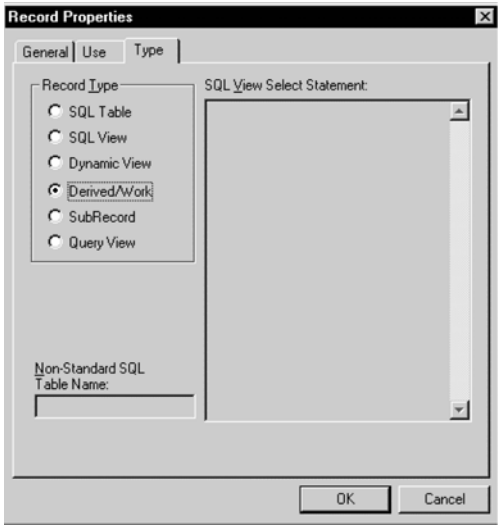


Figure 23.20
Specifying the proper record type for our custom work record

Figure 23.21 The MY_WORK record is created as a Derived/Work record

After filling in the record description in the General tab, we save our new record, and it is automatically added to our project (figure 23.21).

Since the Derived/Work records are usually used as placeholders for different PeopleCode events, there is no need to create a database level table.

23.3.3 Adding Derived/Work fields to our panel

Our next step is to add the newly created derived fields to our panel. We start with the MY_DELETE_EVENT field and place it on the panel as shown in figure 23.22.

Let's also add the MY_DELETE_PROCESS field to our panel. This field is used to initiate the deletion process. This time, we use a push button field type as shown in figure 23.23.

As discussed in part 2 of this book, command push buttons are associated with a Record.Field. When a user presses a push button, the corresponding `FieldChange` PeopleCode event is triggered.

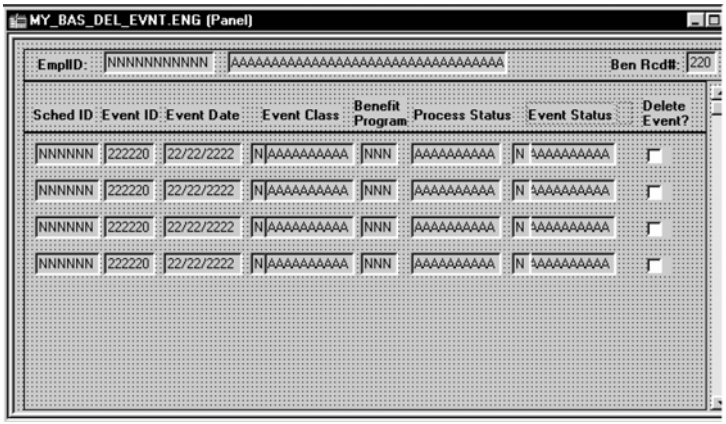


Figure 23.22 Adding Derived/Work field to a panel

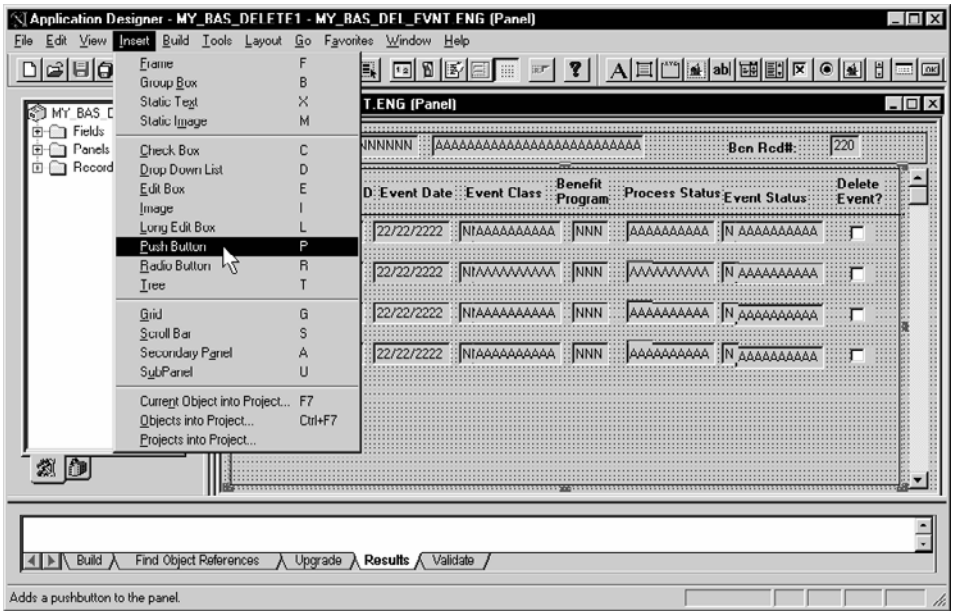


Figure 23.23 Adding a push button field to the panel

We place this field at the bottom of the panel as a stand-alone field, so users can click on it when they decide to execute the `Delete` script. If you look at the panel in figure 23.24, you'll notice that the newly added button looks a little awkward: it has multiplied into four copies.

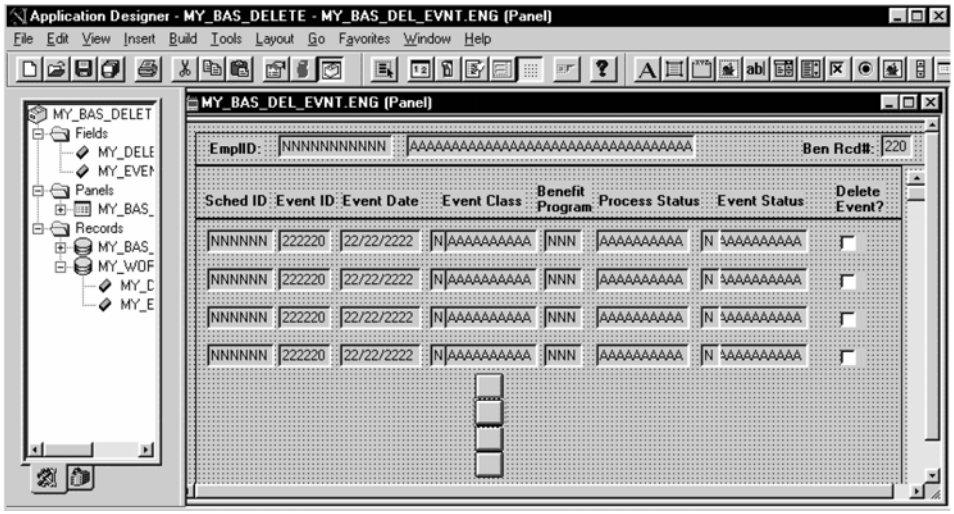


Figure 23.24 The new push button field looks awkward

The reason why we have the four push buttons instead of one is that this field now belongs to the scroll bar area with four occurrences. Any field that belongs to this scroll bar area will be shown in as many occurrences, as defined in the scroll bar properties. Before linking the push button field to our Derived/Work record, let's open the panel order and move the field out of the scroll bar area.

TIP When creating a push button for an entire panel, be certain it is placed outside of the scroll bar with multiple occurrences.

Let's select the highlighted field and move it to the Level 0 area of the panel (figure 23.26.

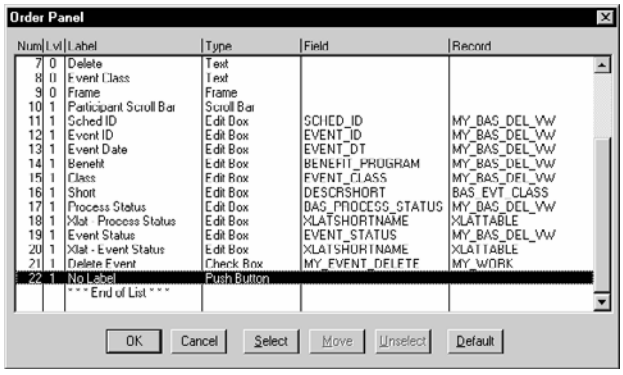


Figure 23.25
At this moment the new push button field belongs to the scroll bar level 1 area

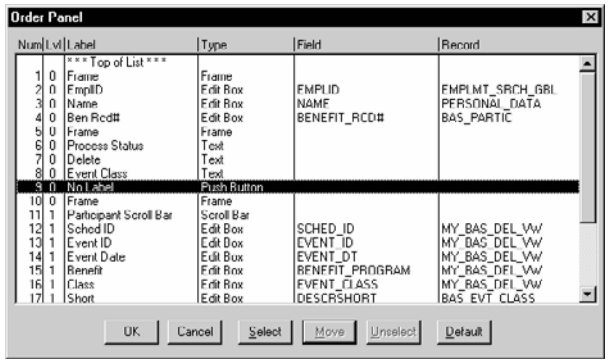


Figure 23.26
Moving the push button field to Level 0

After clicking on the OK button, our panel looks as shown in figure 23.27.

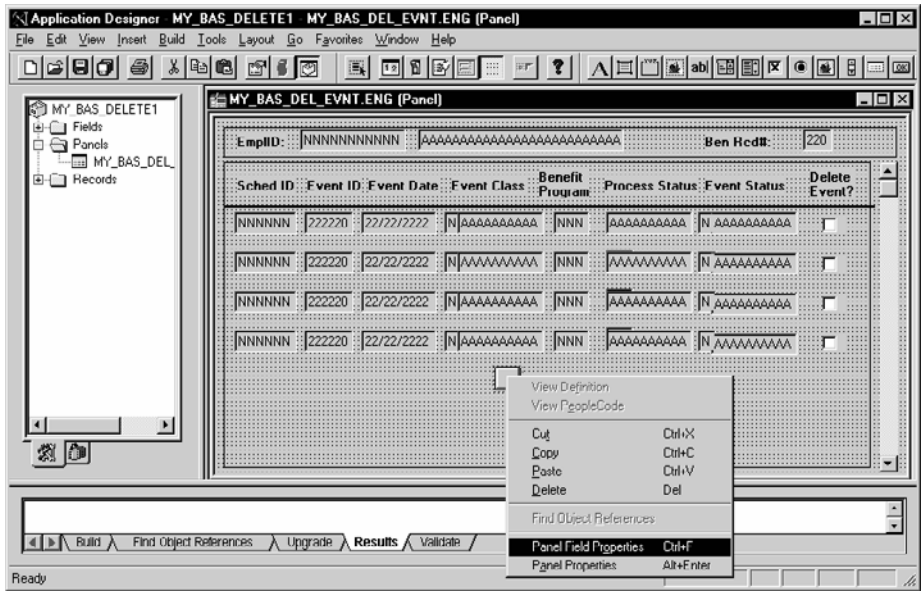


Figure 23.27 Now the push button has been moved out of the scroll bar area

Now we need to set up the push button field properties. This is a special field. It allows you to execute a command or process or call a secondary panel. We already discussed the usage of push buttons in part 2 of this book. Since we will be using this field in order to execute our PeopleCode `Delete` script, the type of push button must be specified as Command. When the user presses the push button, the PeopleSoft Application Processor automatically triggers the PeopleCode script associated with the `FieldChange` event for the MY_DELETE_PROCESS field (figure 23.28). After all the panel's setup is done, we can work on creating our PeopleCode event scripts.

In this panel, we also link our Derived/Work record MY_WORK and the MY_DELETE_PROCESS field with the push button. Let's switch to the second panel in this panel group and specify the label for our new field (figure 23.29).

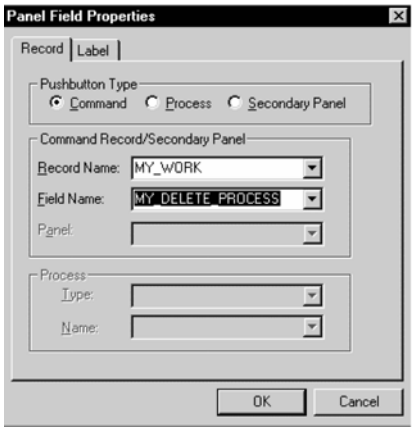


Figure 23.28 Specifying the push button type as Command

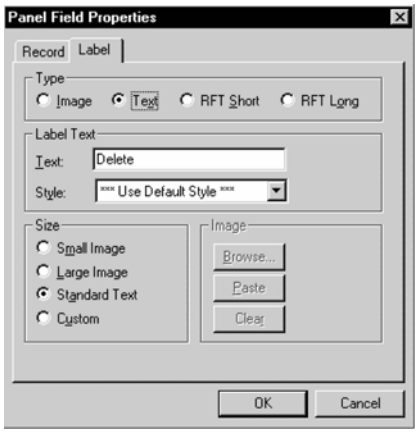


Figure 23.29 Specifying a label for our push button

Note that we used `Text` as the label type for our push button and specified it as `Delete`.

TIP PeopleSoft recommends using short names for push buttons. The long description of the field itself will be shown as a push button tools tip.

After clicking on the OK button, our panel looks as shown in figure 23.30.

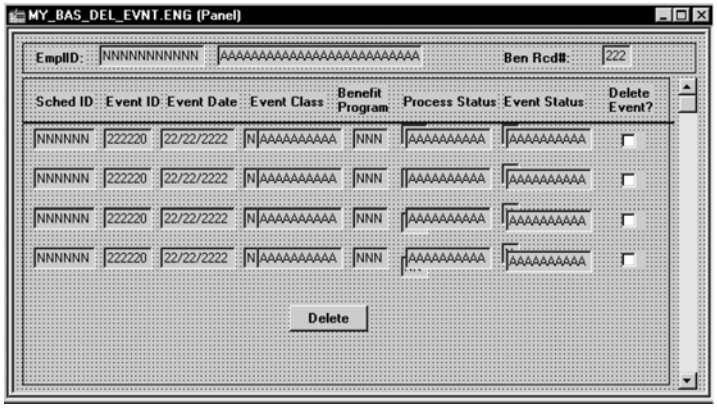


Figure 23.30 Push button Delete is added to the panel

It looks as though we now have all the fields we need. We can test the panel once again in the test mode, to be certain that all fields are aligned, but in order to comprehensively test our new panel, it should be placed into a panel group and menu.

23.4 CREATING A CUSTOM PANEL GROUP

Let's select File →New →Panel Group from the Application Designer and add our panel to the panel group (Insert →Panel) (figure 23.31).

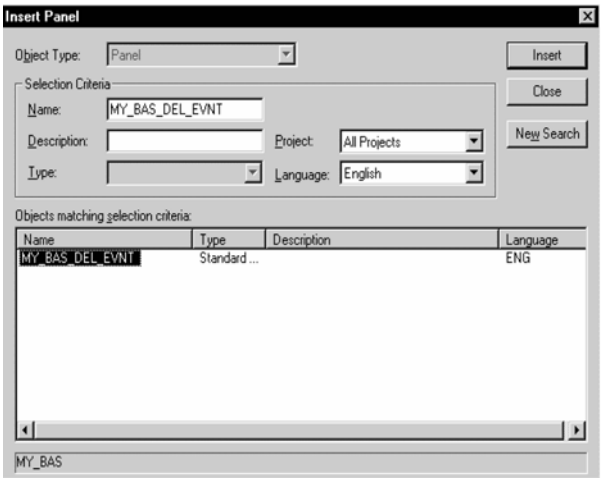


Figure 23.31
Adding the MY_BAS_DEL_EVNT panel to a panel group

After the panel is added, don't forget to click on the Close button. You get a panel group with the default `Item Name` and `Item Label`. These values may be changed. Let's set the `Item Label` to `Event Delete` as shown in figure 23.32.

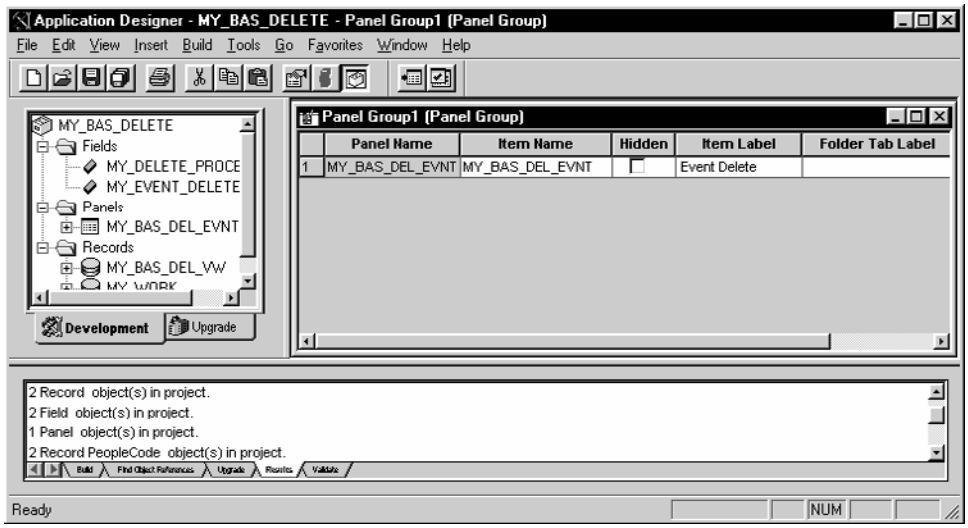


Figure 23.32 Specifying the Item label in the new panel

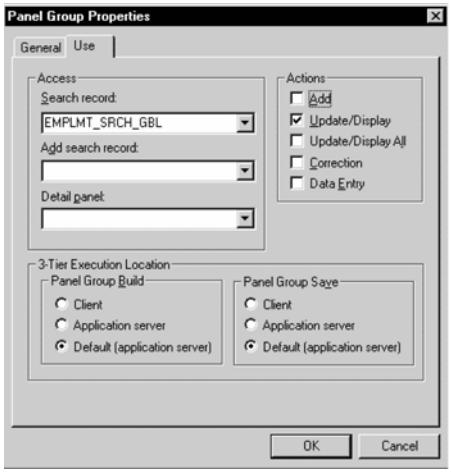


Figure 23.33 Specifying panel group properties

Our next step is to set the panel's properties. Click on the tool bar button and fill in the Panel Group Description and Comments fields in the General tab of the Panel Group Properties panel (figure 23.33). Switch to the Use tab and select the proper search record. Since our header information on the panel is the employee's information, it makes sense to select the EMPLMT_SRCH_GBL view as our search record.

Now we can save the panel group as MY_BAS_DEL_EVNT (figure 23.34).

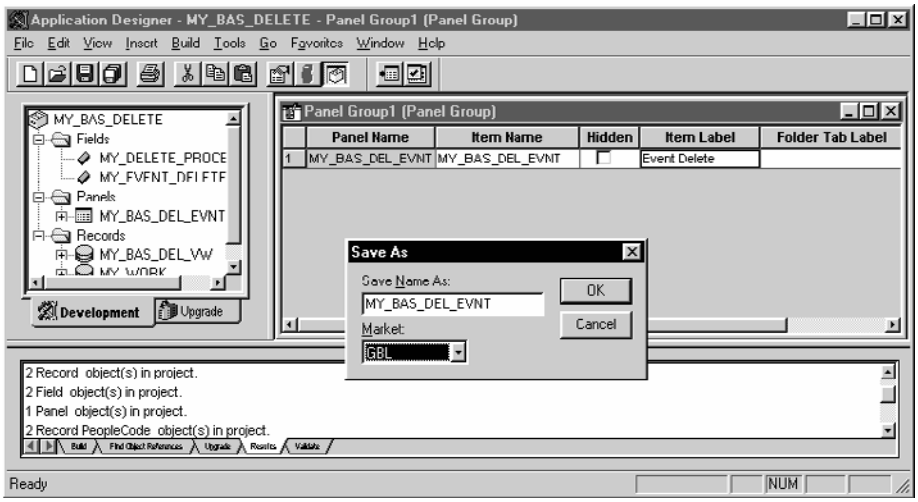
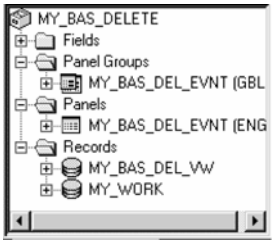


Figure 23.34 Saving the new panel group as MY_BAS_DEL_EVNT



After our new panel group is saved, the MY_BAS_DELETE project looks like that shown in figure 23.35.

Figure 23.35 The new panel group is added to our project

23.5 MODIFYING A MENU

Since our new panel is a part of Benefits Administration, we attach the new custom panel group to the Administer Automated Benefits menu.

Let's open Menu →Administer Automated Benefits. Click on the Process menu bar and select an empty rectangle (figure 23.36).

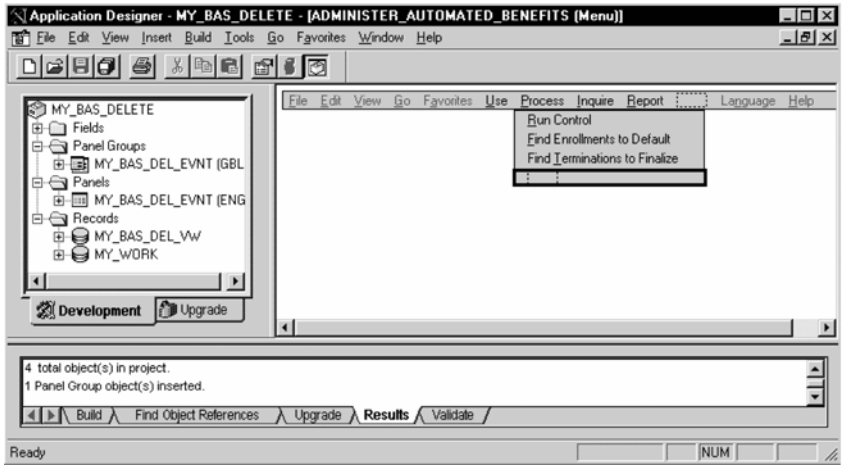


Figure 23.36 Selecting a menu to customize

Double-click on the empty rectangle and specify the new menu item properties (figure 23.37).

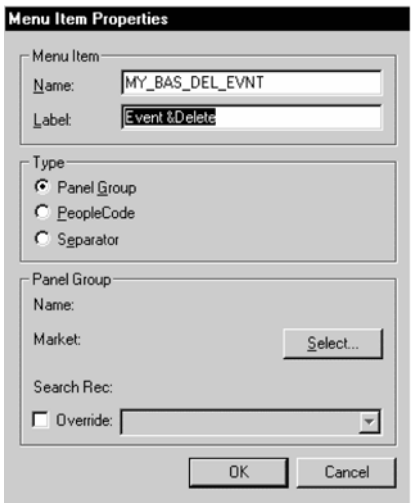


Figure 23.37
Specifying menu item properties

Click on the Select button to attach our custom panel to the new menu item.

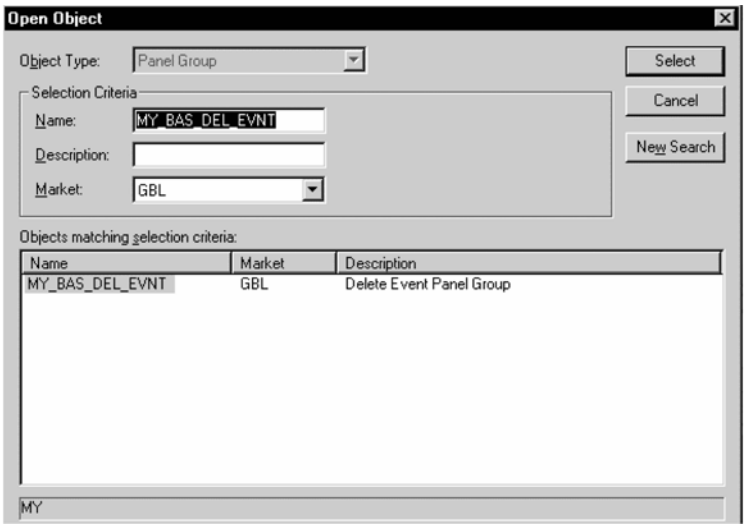


Figure 23.38 Selecting the proper panel group for a new menu item

After clicking on the Select button, the new menu item is inserted into the menu as shown in figure 23.39.

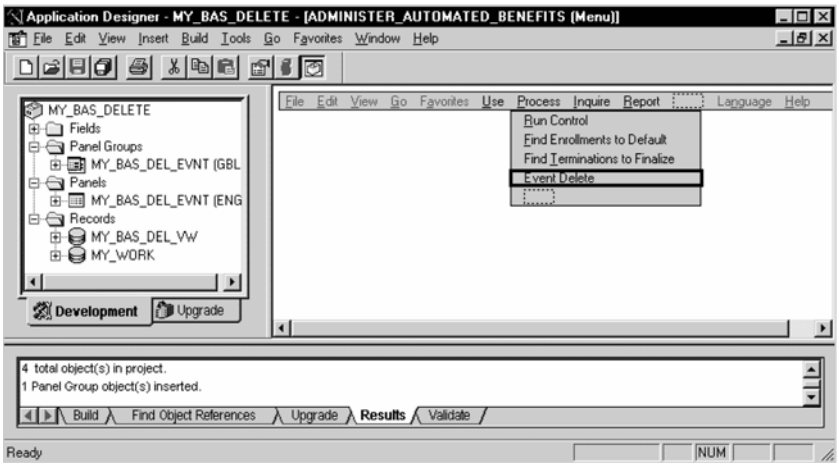


Figure 23.39 The Event Delete menu item is added to the Administer Automatic Benefits menu

When the menu is saved, it is automatically added to our project.

WARNING You can only add an entire menu to your project and not an individual menu item.

When the upgrade project gets executed, let's say, to migrate your modifications to production, the entire Administer Automated Benefits menu in the production database will be replaced with the menu from your development environment. If your development environment is not in synch with production, or concurrent modifications are being done to the menu, this exercise may be dangerous because your entire menu will overlay the production version. Of course, if you execute the Compare and Report process before migration, such an exercise would identify the differences.

23.6 ADDING A PEOPLECODE SCRIPT

Our customizations are almost done, and you may be wondering how we are actually going to delete the selected Benefits Administration events. Remember that we just created a Derived/Work record and added the two fields from this record to our custom panel. Now is the time to write a PeopleCode script that will perform the deletion of the marked records.

Let's open the MY_WORK record by double-clicking on the record in the project and view the PeopleCode events by clicking the tool bar button, or selecting View →View PeopleCode from the Application Designer Menu (figure 23.40).

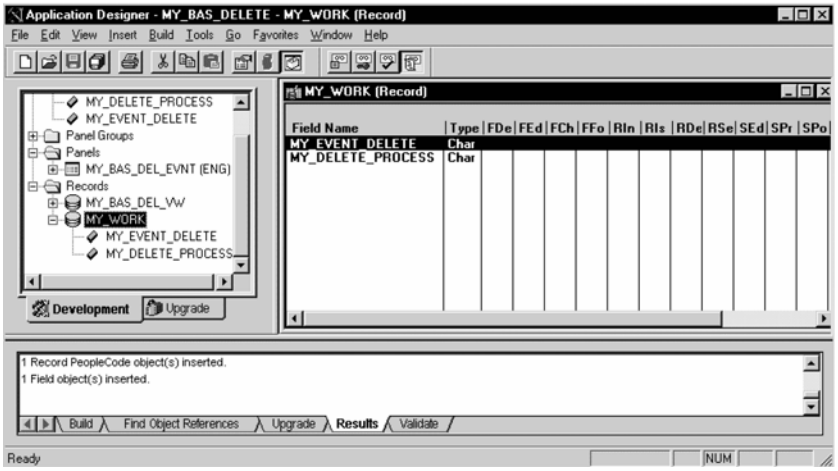


Figure 23.40 The PeopleCode events of the MY_WORK record

We have to accomplish two tasks here. The first is to allow a preliminary selection of events for the potential deletion of all records associated with each selected event. The second task is the actual deletion of the events marked by our users.

Let's start with the first task. Recall that only nonfinalized events can be deleted. In order to distinguish between finalized and non finalized events, we need to check the BAS_PROCESS_STATUS field of the MY_BAS_DEL_VW record. We'll place our code into the `RowInit` event of the MY_EVENT_DELETE field. The PeopleCode program shown in figure 23.41 should be sufficient to perform the task.

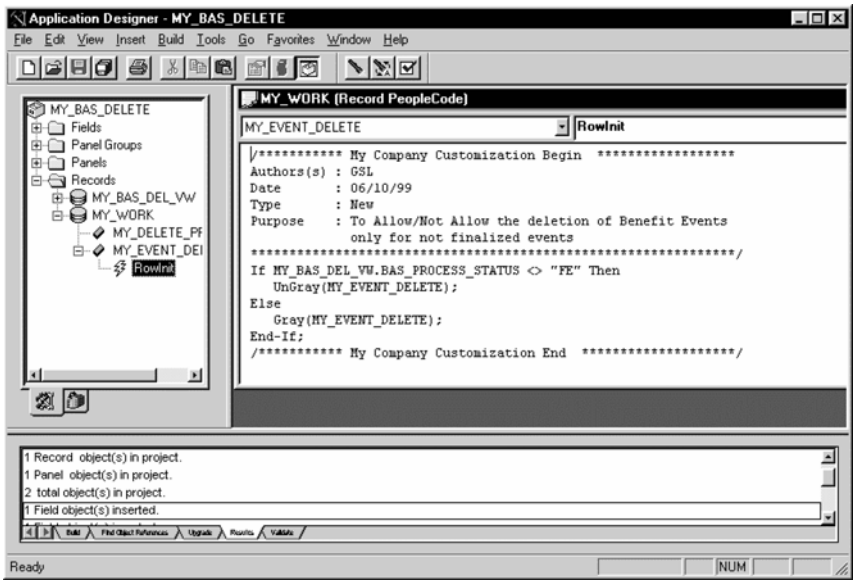


Figure 23.41 Adding a PeopleCode to the RowInit event

As you can see from figure 23.41, this simple code is saved and added to our project. It checks if the BAS_PROCESS_STATUS is equal to 'FE' (`Finalized-Enrolled`) and based on the result of the comparison, it either grays out (disables) or makes the Event Delete field available to users.

Our second task is a bit more complex. When users select an event for deletion, we have to delete all records associated with this event (e.g., all rows added to the database when the event was created). Therefore, detailed knowledge of the Benefits Administration module is necessary. Assuming that we, as developers, are familiar with this product, our next PeopleCode program will look like this:

```
For &ROW = ActiveRowCount(MY_BAS_DEL_VW.EMPLID) To 1 Step - 1
   &DELETE = FetchValue(MY_EVENT_DELETE, &ROW);
   &SCHED_ID = FetchValue(MY_BAS_DEL_VW.SCHED_ID, &ROW);
```

```
   &EMPLID = FetchValue(MY_BAS_DEL_VW.EMPLID, &ROW);
   &BENEFIT_RCD# = FetchValue(MY_BAS_DEL_VW.BENEFIT_RCD#, &ROW);
   &EVENT_ID = FetchValue(MY_BAS_DEL_VW.EVENT_ID, &ROW);
   If &DELETE = "Y" Then
      DeleteRow(RECORD.MY_BAS_DEL_VW, &ROW);
      SQLExec("delete from ps_bas_partic  where sched_id = :1 and emplid =
:2 and benefit_rcd# = :3 and event_id = :4", &SCHED_ID, &EMPLID,
&BENEFIT_RCD#, &EVENT_ID);


     SQLExec("delete from ps_bas_partic_plan where sched_id = :1 and emplid
= :2 and benefit_rcd# = :3 and
event_id = :4", &SCHED_ID, &EMPLID, &BENEFIT_RCD#, &EVENT_ID);
     SQLExec("delete from ps_bas_partic_optn where sched_id = :1 and emplid
= :2 and benefit_rcd# = :3 and event_id = :4", &SCHED_ID, &EMPLID,
&BENEFIT_RCD#, &EVENT_ID);
     SQLExec("delete from ps_bas_partic_cost where sched_id = :1 and emplid
= :2 and benefit_rcd# = :3 and event_id = :4", &SCHED_ID, &EMPLID,
&BENEFIT_RCD#, &EVENT_ID);
     SQLExec("delete from ps_bas_partic_dpnd where sched_id = :1 and emplid
= :2 and benefit_rcd# = :3 and event_id = :4", &SCHED_ID, &EMPLID,
&BENEFIT_RCD#, &EVENT_ID);
     SQLExec("delete from ps_bas_partic_invt where sched_id = :1 and emplid
= :2 and benefit_rcd# = :3 and event_id = :4", &SCHED_ID, &EMPLID,
&BENEFIT_RCD#, &EVENT_ID);
   End-If;
End-For;
```

Let's examine our PeopleCode. Our goal here is to delete records from the following BenAdmin tables:

- PS_BAS_PARTIC
- PS_BAS_PARTIC_PLAN
- PS_BAS_PARTIC_OPTN
- PS_BAS_PARTIC_COST
- PS_BAS_PARTIC_DPND
- PS_BAS_PARTIC_INVT

Since we need to delete only records selected by our user, the first part of the code selects the MY_EVENT_DELETE field from every row (in the `For` loop) and checks if it is marked by users for deletion. It also fetches key fields such as SCHED_ID, EMPLID, BENEFIT_RCD#, and EVENT_ID for each record and uses these fields as bind variables in the database `Delete` statements.

Please note that we are performing the deletions by using the SQLExec commands because there are a multitude of tables that contain information for each Benefit event.

Now we need to make a decision on where to place this PeopleCode program. In other words, what PeopleCode event should the program belong to? The first

thought would be to place it in the FieldChange event of the MY_DELETE_PROCESS record. After all, we created the push button field in the panel to do just that. We also know that push button commands should have their processes in the FieldChange event, because this event is triggered when the button is pushed. But if you recall our PeopleCode discussions in part 3 of this book, since our PeopleCode contains the SQLExec statements, they could only be issued in the SavePreChg, WorkFlow, or SavePostChg events. In order to resolve this conflict, let's use a trick here. As soon as the push button is activated, it will trigger the PeopleCode from the FieldChange event (figure 23.42).

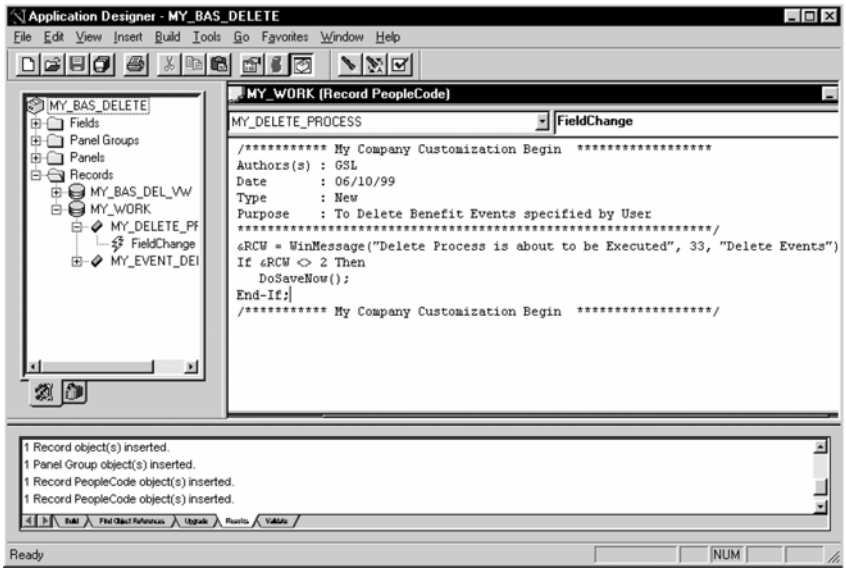


Figure 23.42 The FieldChange event PeopleCode script executed when the Delete push button is clicked on.

In the PeopleCode shown in figure 23.42, we first provide our users with a facility to cancel the Delete request by prompting them to accept or cancel their request. It's always a good idea to give your users a second chance to decide if they really want to delete the records. If they decide to go on, we execute the PeopleCode DoSaveNow() function. This function, in turn, triggers the execution of all Save events. Therefore, if we place our PeopleCode program described in the previous page into the SavePostChg event, it will be executed immediately after the user clicks on the OK button of the prompt box.

Let's select the SavePostChg event (figure 23.43).

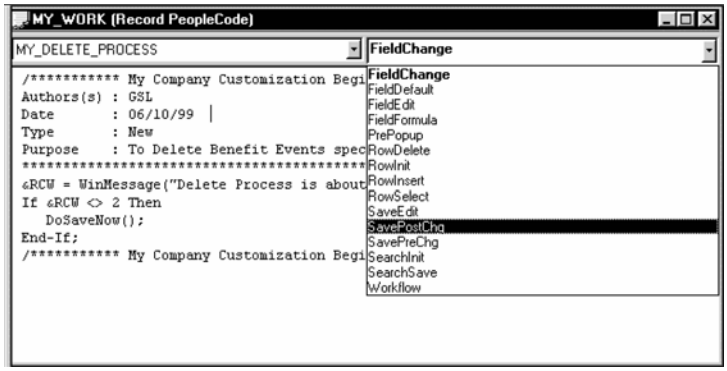


Figure 23.43 Selecting the SavePostChg event

Our next step is to copy the PeopleCode we created earlier and paste it into the `SavePostChg` event. We also add some comments as a standard header (figure 23.44).

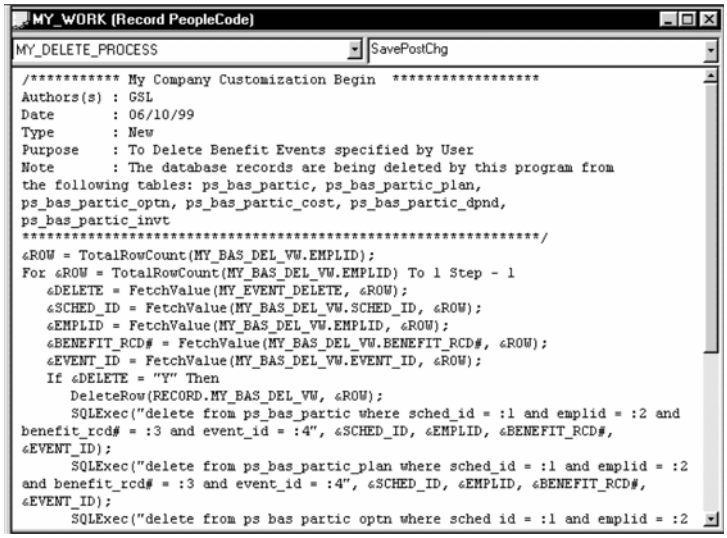


Figure 23.44 Adding a PeopleCode program to delete all selected events.

After saving the PeopleCode script and adding it to our project, we just need to grant security access to our users and ourselves before testing.

23.7 GRANTING SECURITY ACCESS

Navigation: Go →PeopleTools →Security Administrator →Open →ALLPANLS →Menu Items

Select the ADMINISTER_AUTOMATIC_BENEFITS menu.

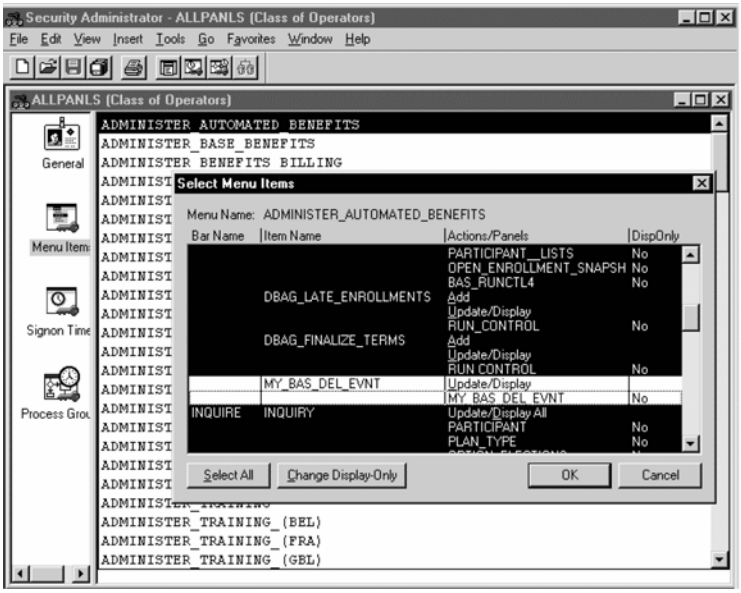


Figure 23.45 Granting security access to MY_BAS_DEL_EVT menu item

Highlight the two lines that belong to the MY_BAS_DEL_EVNT menu item, press OK, and save the security changes for operator class ALLPANLS. Repeat the same steps for your BenAdmin user's access.

Now is the time for a real test.

23.8 TESTING OUR CHANGES

Navigation: Go →Compensate Employees →Administer Automated Benefits → Use →Event Delete

Let's select the same employee ID, 8845, as we did at the beginning of our chapter.

Our new Event Delete panel appears showing all BenAdmin events associated with the selected employee. When scrolling through the records, we can see that for all `Finalized/Enrolled` events (Process Status= `Enrolled`) our Delete Event checkboxes are correctly grayed out, not allowing users to delete these events. Three events remain available for deletion: VN97, VN98, and VNBEM. Click on the Delete Event checkbox for these three open events, then click on the Delete push button as shown in figure 23.46.

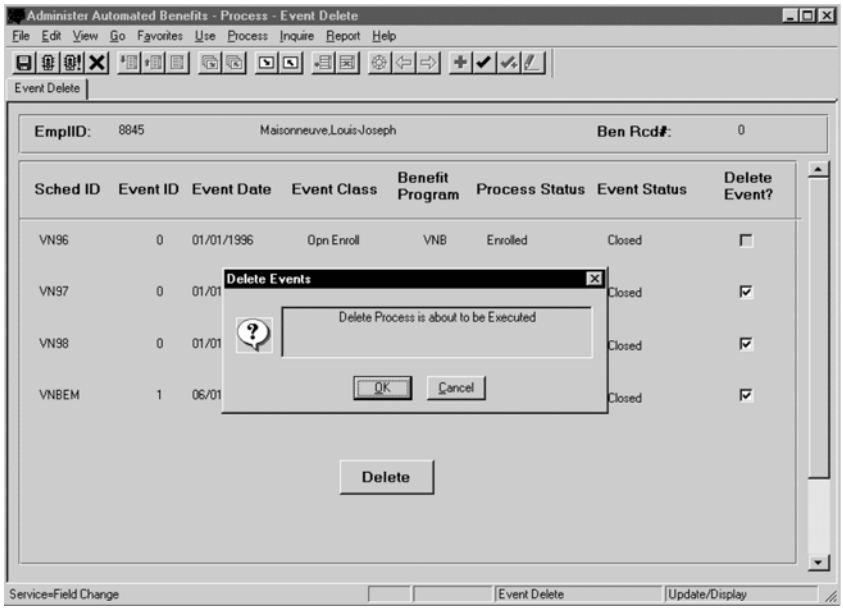


Figure 23.46 Deleting three open events

So far, the PeopleCode that we placed in our events worked just fine. We've got a prompt that gave us an opportunity to cancel the Delete process if necessary. Let's say we want to go ahead and click on the OK button. The results of our `Delete` script execution from the `SavePostChg` PeopleCode event will be as displayed in figure 23.47.

We see only two events now. These two events were not targeted for deletion. It looks like all our changes are working. Since our modification involved a database update, it is necessary to verify if the records have been actually deleted from the tables. Using database tools such as SQLPlus for Oracle or SQL Talk in SQL Server or any other tool that is available in your environment, you can simply select the information from the tables specified in our `Delete` PeopleCode script and make certain that the records are not there.

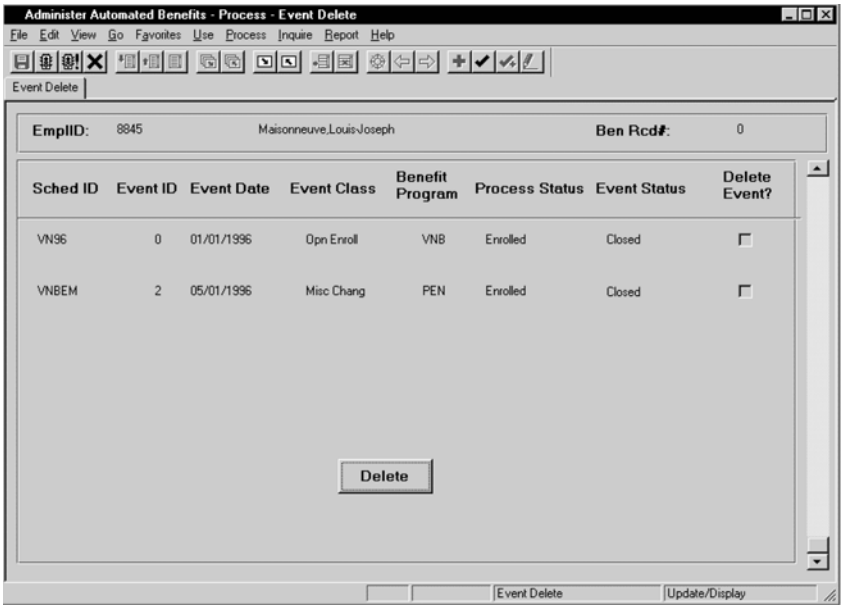


Figure 23.47 All three selected events have been deleted from the screen and from the database

23.9 POSSIBLE IMPACT ON FUTURE UPGRADES

In this chapter we illustrated a classical example of enhancing user functionality by using the "Add" as opposed to the "Modify" approach. During our development process we were trying to minimize the impact on future upgrades. We developed our own custom records, panel, PeopleCode programs, panel group, and menu item. As in the previous chapter, we used distinctive names for all of our objects, documented our changes, and created a project to keep track of all customizations. The only modified PeopleSoft-delivered object was the Administer Automated Benefits menu to which we added our newly created menu item, linking it to our custom panel group.

Still, as we have already discussed, there is always a possibility that PeopleSoft delivers the similar or same functionality. In this case, you have to evaluate the PeopleSoft changes. If PeopleSoft's new features are in fact similar to ours, they should take precedence over our own.

KEY POINTS

1. You can greatly improve user productivity by extending PeopleSoft-delivered functionality
2. When creating your panel by cloning a delivered panel, make certain that all fields in your new panel belong to your records. Always check the Order panel carefully.
3. The Derived/Work records are often used to display temporary values on a panel. The fields in such records are ideal placeholders to store and trigger PeopleCode programs.
4. Command push buttons are associated with a record.field. When a user presses on a push button, the corresponding Field Change PeopleCode event is triggered.
5. When your panel has a functionality to delete data from database tables, it is a good idea to give your users an option to confirm or cancel the deletion.

CHAPTER 24

Customizing security search records, PeopleCode, and menus

In the previous chapter, we discussed the customization of PeopleSoft-delivered objects and creation of new objects such as fields, records, panels, panel groups, and some PeopleCode scripts. In the next example, we will demonstrate how to change the behavior of PeopleSoft's record selection mechanism by simply modifying the delivered search record. Please keep in mind that this exercise requires a good understanding of PeopleSoft's department security and security search records.

Let's turn to exercise 4:

> Allow users to access records of employees transferred to another department.

Many PeopleSoft HRMS developers have often heard their users in HR departments complain that they can not access the records of their former employees who had been transferred to different departments. In fact, often a business reason exists for such requests. Of course, security must be in place to prevent users from having access to unauthorized information. Consequently, an HR manager can have access to the records of only those employees who belong to the department to which the HR manager is presently assigned. In this exercise, we need to find a way to allow HR managers access to the records of their former employees assigned to different departments. First, let's identify the objects that need to be customized or created.

24.1 WHAT OBJECTS SHOULD BE CUSTOMIZED OR ADDED?

Let's refresh our knowledge on what exactly prevents users from seeing their former employee's records.

As discussed in part 2 of this book, some panels in PeopleSoft are accessed with the help of special search views. These views are designed to restrict the user's access to unauthorized data. In HRMS, for example, employee information is protected based on the Department Security delivered by PeopleSoft.

When we select a particular employee's record by entering full or partial information in the search box, the PeopleSoft Application Processor builds a `Select` statement according to the information entered by the user, and based on the search record specified for the particular Panel Group. Our goal, therefore, is to look at the security search record and figure out how to modify it.

In order to understand how PeopleSoft's Application Processor works, and what exactly happens when a user enters the particular selection criteria, let's use the PeopleSoft `Trace` utility.

First we need to check if the trace filename is specified in the Configuration Manager panel (figure 24.1).

After the trace is activated, the system will write the trace file to the specified directory. In our case the directory for the trace output file is specified by the environment variable `%TEMP%`. The default filename for the trace file is DBG1.tmp. You can verify the settings for your Windows `Temp` directory in the Process Scheduler tab of the Configuration Manager.

Navigation: Edit →Preferences →Configuration

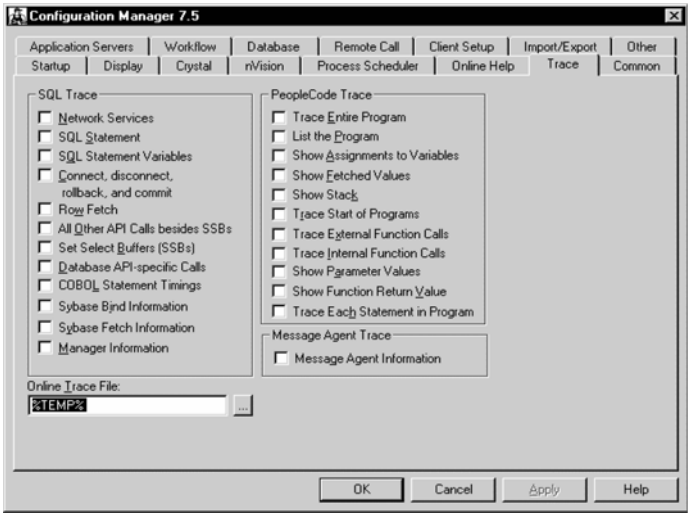


Figure 24.1 Verifying the Trace filename

Our next step is to activate the trace (figure 24.2).

Navigation: Go →PeopleTools →Utilities

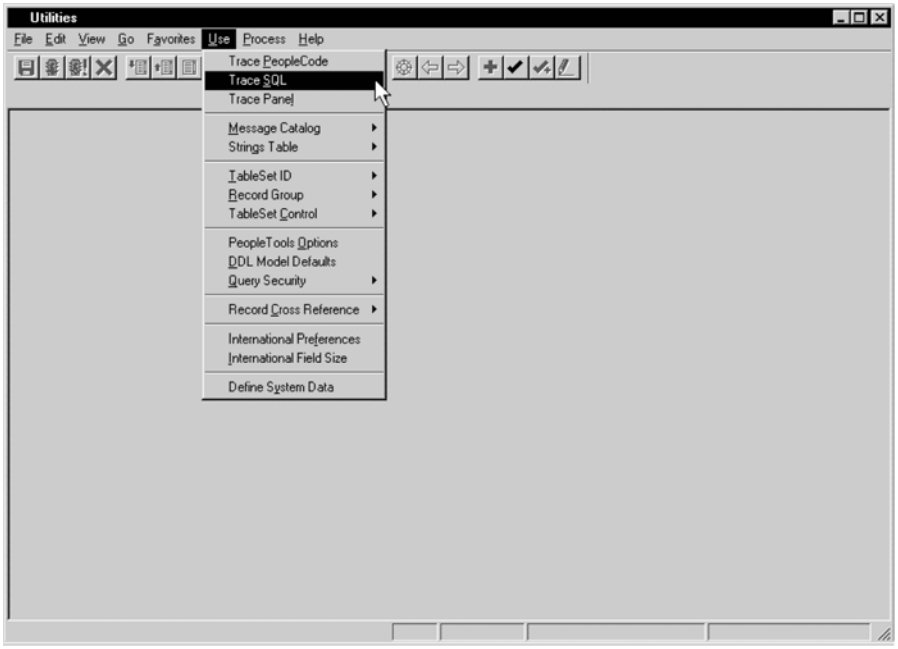


Figure 24.2 Selecting the Trace SQL utility

From the Utilities Menu, select Use →Trace SQL.

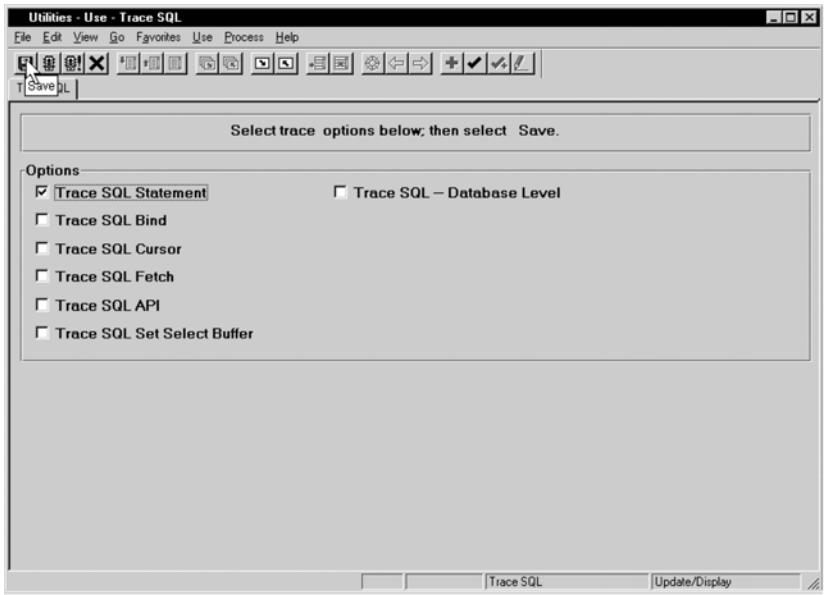


Figure 24.3 The Trace SQL Utility panel

On the panel (in figure 24.3), we can select any combination of traces we need. For our purposes, we select only the first one since we only need to see the SQL statement that Application Processor constructs based on the parameters in our request. Make sure you save the selection. This panel is a bit misleading. When you first open it, the panel already has the `Trace` SQL statement checked on. This is the panel default option. The trace will not be activated until you actually save the selection. Please note that after you activate the trace, all your following steps will be recorded in the trace file.

TIP In order to make the trace file reasonably small, activate the trace right at the point where you need it and stop the trace as soon as your testing is over. It's always easier to work with smaller files.

After saving the trace options, we minimize the Utilities panel and go through a couple of steps that will provide the information we are looking for.

We type "Smith" as our search criteria name and press OK (figure 24.4).

At this particular moment, we already have all the requested records selected. We should stop the trace, then examine our trace log file. In order to stop the trace, maximize the Utilities panel, unclick the trace selections, then save the panel (figure 24.5).

Navigation: Go →Administer_Workforce_(U.S.) →Use →Job Data → Update/Display All

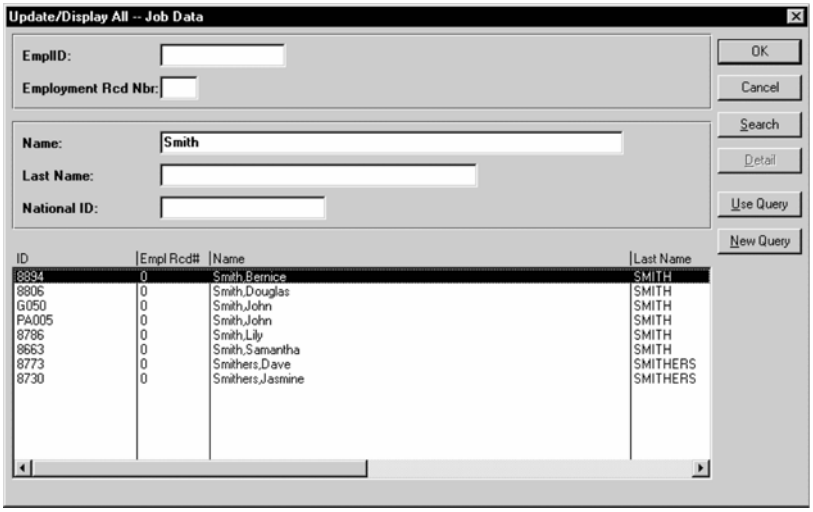


Figure 24.4 Selecting Smith with a trace activated behind the scene

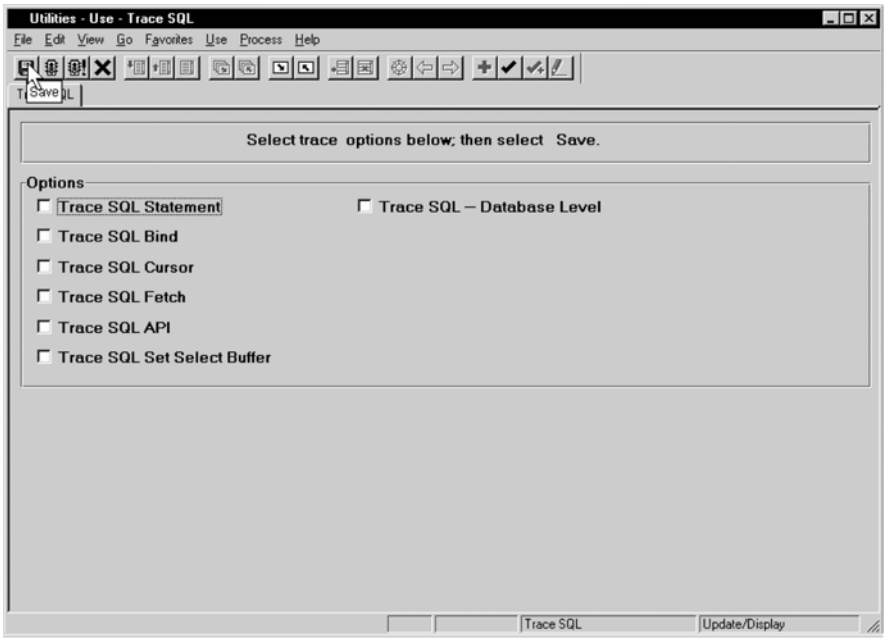


Figure 24.5 Deactivating the Trace utility

TIP Do not forget to deactivate the Trace utility as soon as the test is done.

Now we can take a look at our trace file.

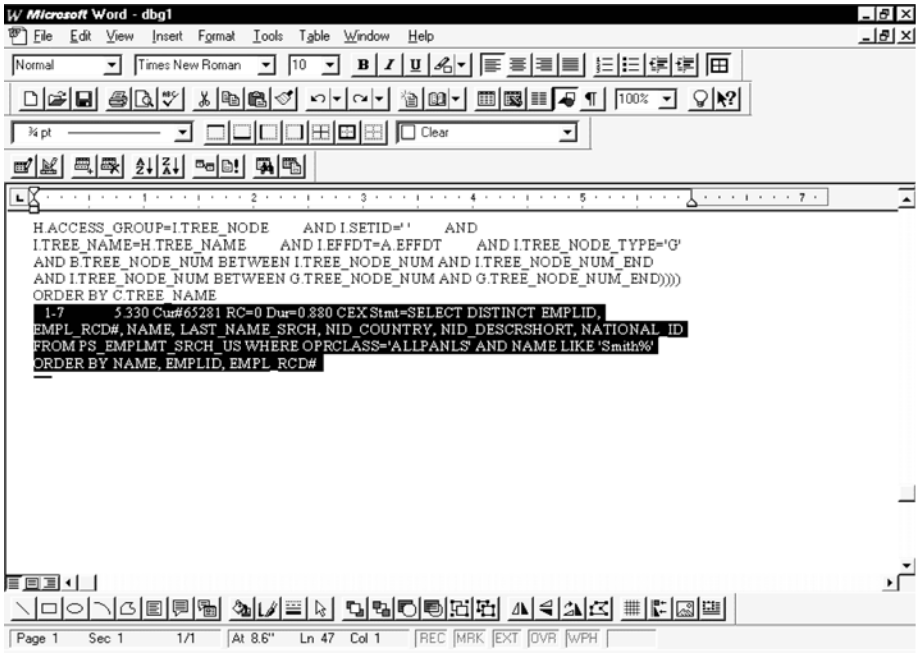


Figure 24.6 A portion of the Trace file

Since we want to see how the system selects the records with the last name starting with "Smith," let's find the corresponding SQL statement in our file. It should be either the last one or close to the end, since we stopped our trace as soon as the selection was done.

If we cut the statement from the trace file, paste into Notepad, and reformat it for readability purposes, the PeopleSoft's `Select` statement looks like the following:

```
SELECT DISTINCT EMPLID, EMPL_RCD#, NAME, LAST_NAME_SRCH, NID_COUNTRY,
NID_DESCRSHORT, NATIONAL_ID
FROM PS_EMPLMT_SRCH_US
WHERE OPRCLASS='ALLPANLS'
AND NAME LIKE 'Smith%'
ORDER BY NAME, EMPLID, EMPL_RCD#
```

As you can see, the `Select` statement built to get records from the database looks pretty simple. The records are selected from the PS_EMPLMT_SRCH_US table, which is a search record specified for the Job Data panel group. This tells us that this record is responsible for selecting the requested information. The selection is limited to the search criteria ("Smith") and the operator class (ALLPANLS). Since PS_EMPLMT_SRCH_US is a security search view, our `Select` statement only returns the records that the ALLPANLS operator class is allowed to see. Note the `DISTINCT` keyword in the `Select` statement. This way the `Select` returns distinct rows and builds a list box for the operator's further selection. If the operator has an appropriate access to the record, this record is displayed in the list box.

Do we always need to activate the trace in order to figure out what search record is used? No, we demonstrated this as a convenient way to learn what is actually going on behind the scene. The next time you need to deal with search records, you will remember how it works, and the only thing you need to verify then is the security search record used for a particular panel group.

Let's now make certain that the PS_EMPLMT_SRCH_US record is, in fact, the search record for the JOB_DATA panel group. Selections from the PS_EMPMT_SRCH_US record is indicated by ❶.

Navigation: GO →PeopleTools →Application Designer →Open →Pane Group →JOB_DATA

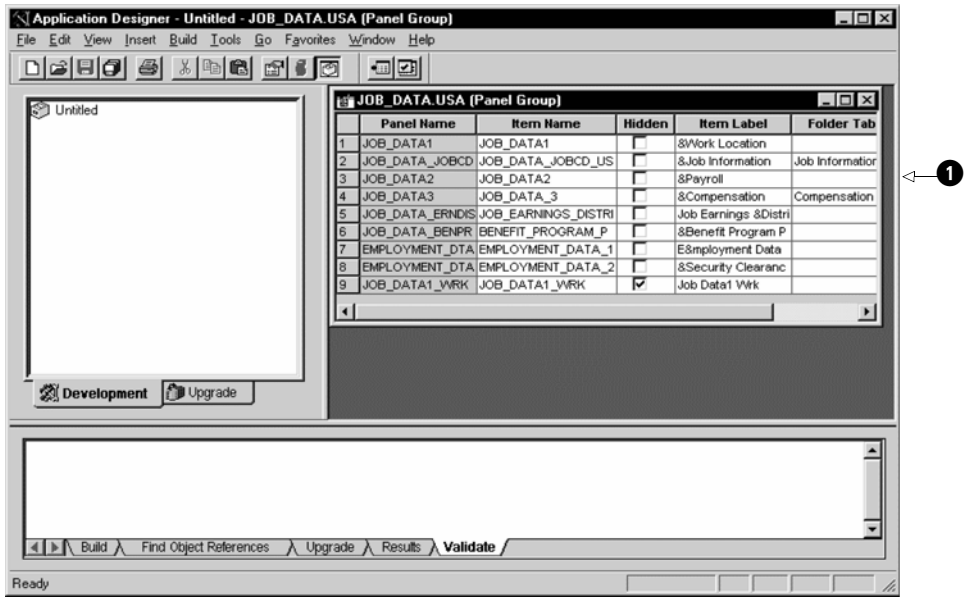


Figure 24.7 Examining the JOB_DATA panel group

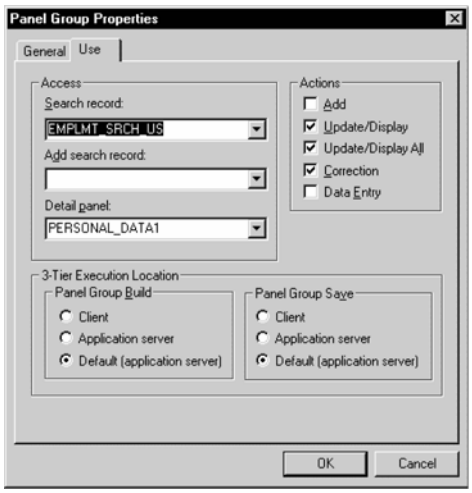


Figure 24.8 Looking for the Search record

In order to find out what search record is attached to this panel group, click on the button or select File → Object Properties.

The screen in figure 24.8 appears.

The search record that is used for the JOB_DATA panel group is the EMPLMT_SRCH_US record. In our next step we will open and examine this search record.

Navigation: GO → PeopleTools → Application Designer → Open → Record → EMPLMT_SRCH_US.

After opening the record, let's take a look at its properties. Click on the button, then select the Type tab.

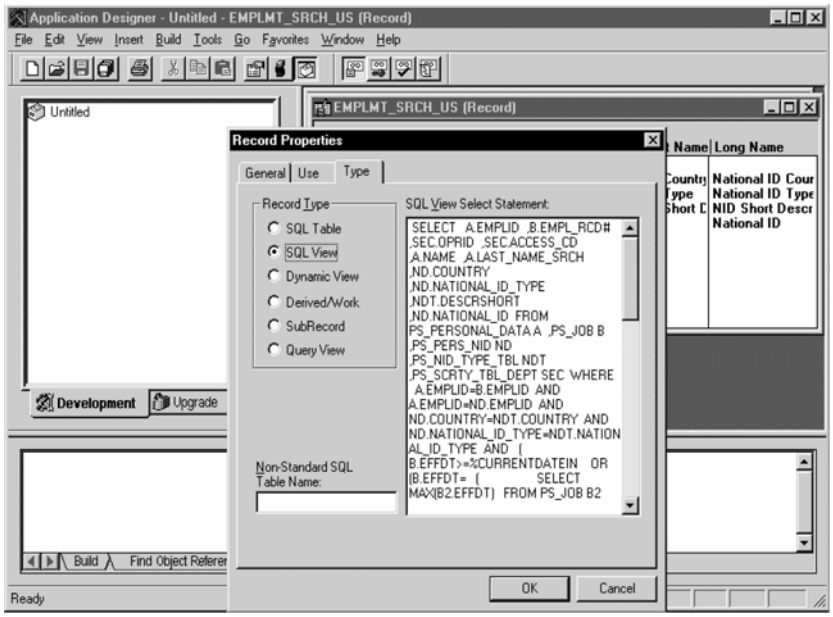


Figure 24.9 The Type tab of the EMPLMT_SRCH_US record properties

Let's copy and paste the SQL view definition, so we can see it better:

Listing 24.1

SQL View Select statement for the EMPLMT_SRCH_US record

```
SELECT A.EMPLID, B.EMPL_RCD#, SEC.OPRID, SEC.ACCESS_CD, A.NAME,
A.LAST_NAME_SRCH, ND.COUNTRY, ND.NATIONAL_ID_TYPE, NDT.DESCRSHORT,
ND.NATIONAL_ID
FROM
PS_PERSONAL_DATA A,
PS_JOB B,
PS_PERS_NID ND,
PS_NID_TYPE_TBL NDT,
PS_SCRTY_TBL_DEPT SEC
WHERE
    A.EMPLID            = B.EMPLID
AND A.EMPLID            = ND.EMPLID
AND ND.COUNTRY          = NDT.COUNTRY
AND ND.NATIONAL_ID_TYPE = NDT.NATIONAL_ID_TYPE
AND (B.EFFDT >= %CURRENTDATEIN
     OR
    (B.EFFDT  =
       (SELECT MAX(B2.EFFDT)
        FROM PS_JOB B2
        WHERE B.EMPLID  = B2.EMPLID
   AND B.EMPL_RCD# = B2.EMPL_RCD#
   AND B2.EFFDT   <= %CURRENTDATEIN)
      AND B.EFFSEQ    =
          (SELECT MAX(B3.EFFSEQ)
           FROM PS_JOB B3
           WHERE B.EMPLID    = B3.EMPLID
             AND B.EMPL_RCD# = B3.EMPL_RCD#
             AND B.EFFDT     = B3.EFFDT )
    )
    )
AND SEC.ACCESS_CD = 'Y'
AND EXISTS
   (SELECT 'X'
    FROM PSTREENODE SEC3
    WHERE SEC3.SETID   = SEC.SETID
    AND SEC3.SETID     = B.SETID_DEPT
    AND SEC3.TREE_NAME = 'DEPT_SECURITY'
    AND SEC3.EFFDT     = SEC.TREE_EFFDT
    AND SEC3.TREE_NODE = B.DEPTID
    AND SEC3.TREE_NODE_NUM BETWEEN
        SEC.TREE_NODE_NUM AND SEC.TREE_NODE_NUM_END
    AND NOT EXISTS
        (SELECT 'X'
         FROM PS_SCRTY_TBL_DEPT SEC2
         WHERE SEC.OPRID  = SEC2.OPRID
         AND   SEC.SETID  = SEC2.SETID
         AND   SEC.TREE_NODE_NUM <> SEC2.TREE_NODE_NUM
```

```
        AND   SEC3.TREE_NODE_NUM BETWEEN
              SEC2.TREE_NODE_NUM AND SEC2.TREE_NODE_NUM_END
        AND   SEC2.TREE_NODE_NUM BETWEEN
              SEC.TREE_NODE_NUM AND SEC.TREE_NODE_NUM_END
        )
)
```

As you may have noticed in listing 24.1, this view definition is fairly complex. Our goal is not to completely redesign this view, but to understand how it can be customized to allow our users access to the required information. This view selects the employee records in departments that a particular operator class is allowed to access. Take a closer look at the view. It selects the latest (EFFDT is in descending order) record from the PS_JOB table, based on the department found in this record and the operator class. What if, instead of selecting the top PS_JOB record, we allow the selection of any record from the department in which the employee used to work in the past or is currently employed? This will allow our users to access employee's records as required. We will perform the actual modifications in the next subchapters. Here, we just have to figure out what objects to modify. Now that we found what object is responsible for the security access, the question is: "Is it safe just to go ahead and change the security view?" And the answer is "Absolutely NOT." This view is used not only in a multitude of panels, but also as a query security record. Changing this view may result in an incorrect panel access as well as inaccurate reporting. Let's find all the objects that use this view.

TIP Use the Find Object References PeopleSoft utility to identify all the on-line objects that might be affected by your customization.

After the record is displayed, click on Edit → Find Object References (figure 24.10).

At the end of the search for object references, PeopleSoft displays the count of objects found. There are 123 objects currently using the EMPLMT_SRCH_US record. Therefore, if you change the EMPLMT_SRCH_US record, all 123 objects will be affected in one way or another.

A better and safer way to perform our customization would be to create a new custom view based on this record and modify it to satisfy the user's requirements.

What about the panel and the panel group? We have to create a custom panel group with the new security search view attached. In addition, a new menu item must be added to the existing menu. Is that all? Let's take a look at our requirements again. Our users would like to have access to the records of their former employees, but they should not see the salary sensitive information for departments to which they are not supposed to have access. How do we do this? PeopleCode program can help us hide salary-related fields.

Navigation: GO →Application Designer →Open →Record →EMPLMT_SRCH_US

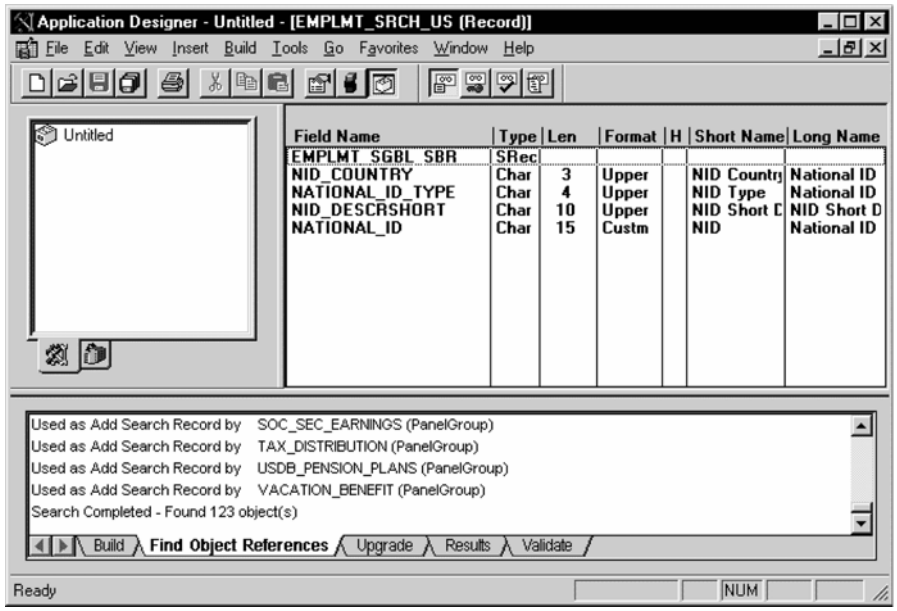


Figure 24.10 Displaying all objects that use the EMPLMT_SRCH_US record

To summarize, we have identified the following objects that must be customized or created:

- a custom security search view
- a panel group with this new view attached
- a menu item
- a PeopleCode script

24.2 CREATING A CUSTOM SECURITY RECORD

Our task is pretty simple. We already learned that our new record should be cloned from the PeopleSoft-delivered EMPLMT_SRCH_US record. Let's open this record and save it as MY_EMPLMT_XFER (figure 24.11).

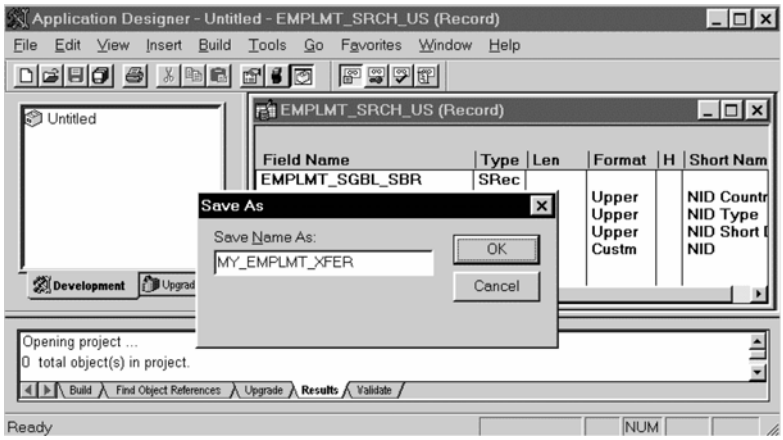


Figure 24.11 Creating the MY_EMPLMT_XFER search view by cloning the EMPLMT_SRCH_US record

When saving the record, we get a warning message asking if the PeopleCode should be copied along with the record (figure 24.12). Since we are planning to modify the existing SQL view definition, let's answer "Yes," and copy all the PeopleCode events.

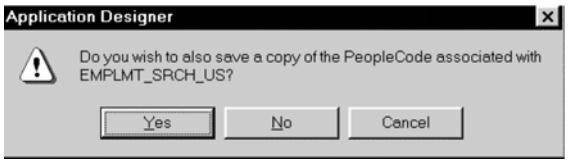


Figure 24.12 Saving PeopleCode programs along with the record definition

Since we just created and saved our new object, it is added to a project. Let's save the project as MY_JOB_XFER (figure 24.13).

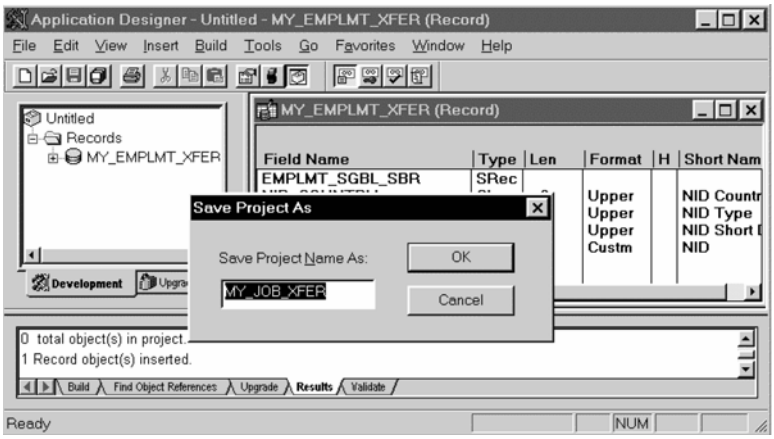


Figure 24.13 Adding the MY_EMPLMT_XFER record to a project

So far, we've created the record definition. Our next task is to modify the actual SQL view definition. Let's press the Alt →Enter and customize our search record.

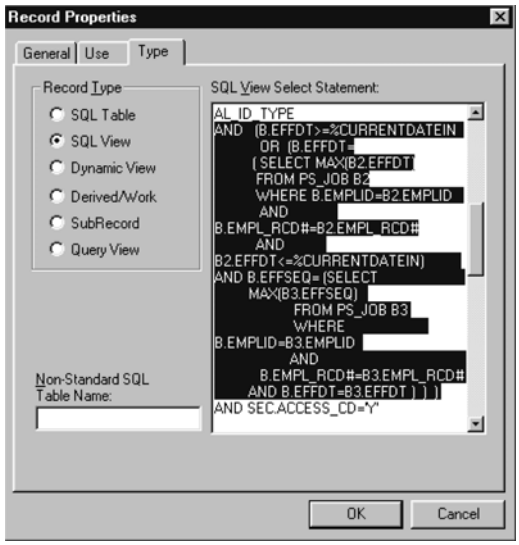


Figure 24.14
An SQL View definition for the EMPLMT_SRCH_US record

As discussed earlier in this chapter, this view is responsible for allowing users access to particular information. The highlighted portion of the view represents the SQL logic that selects the current PS_JOB record. This is exactly a portion of the SQL that we were planning to replace. After deleting the highlighted portion of the `Select` statement, our new view definition appears as shown in figure 24.15.

Figure 24.15
The SQL view after deleting the EFFDT logic

After pressing the OK button to accept our changes, let's switch to the first tab of the panel group and document our modifications (figure 24.16).

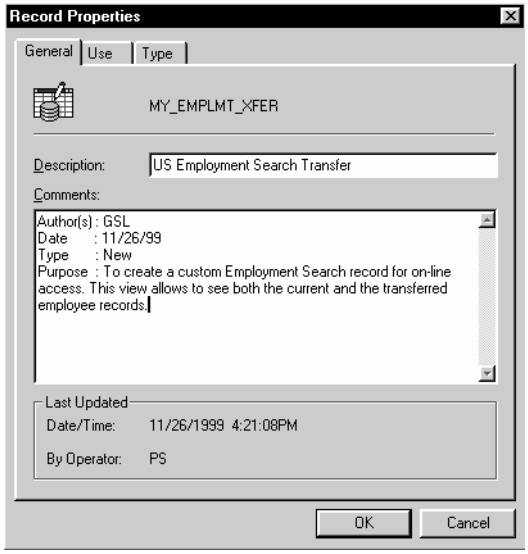


Figure 24.16
Documenting our changes in the General tab of the Record Properties

Our next step is to create a database level view.

Navigation: Build →Current Object

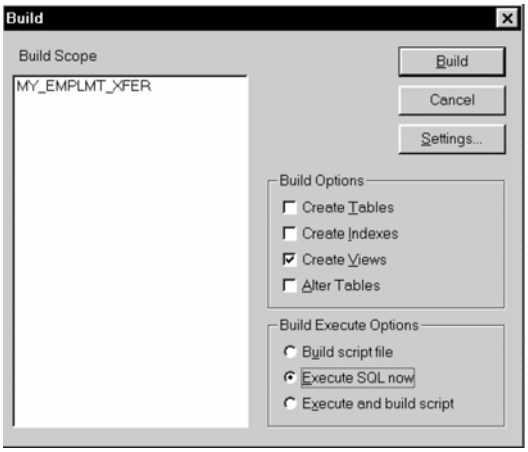


Figure 24.17
Building the database level view

Select Create Views and Execute SQL now options and click on the Build button (figure 24.17).

Our view is ready. We did not get syntax errors which means that the view is valid. Will it work as expected? We'll find this out soon. First, we need to create a new panel group and a new menu item.

24.3 CREATING A CUSTOM PANEL GROUP

As you may have already guessed, we are going to clone an existing panel group in order to create our own. Let's find the name of the panel group used in the Administer_Workforce_US menu (figure 24.18).

We open this menu and click on the Use menu bar.

Figure 24.18 Finding the name of a panel group

If we double-click on the Job Data menu item, we see the panel group name and the search record used for this panel group in the Menu Item Properties panel (figure 24.19).

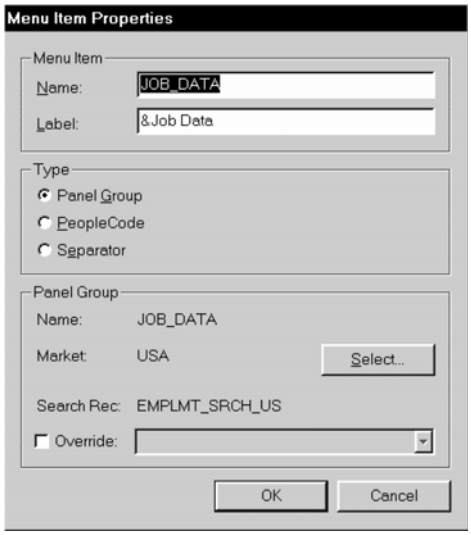


Figure 24.19
The JOB_DATA panel group is used to access the Job Data menu item

Let's open the JOB_DATA panel group and save it as MY_JOB_DATA_XFER (figure 24.20).

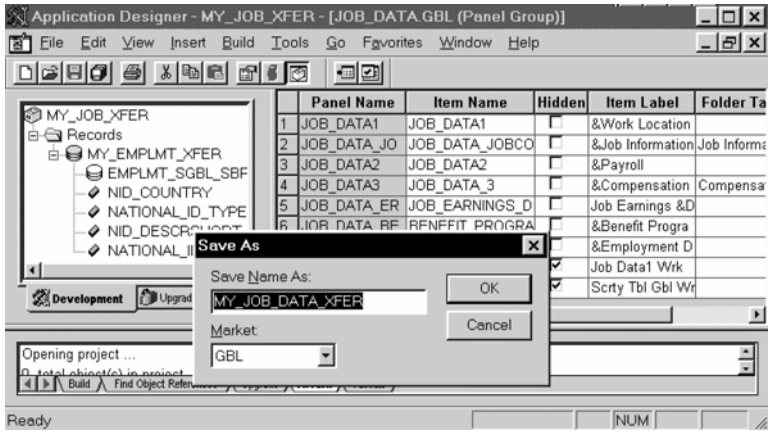


Figure 24.20 Saving the MY_JOB_DATA_XFER panel group

Since we just created another custom object, it is added to our project. The MY_JOB_XFER project now contains two of our custom objects (figure 24.21).

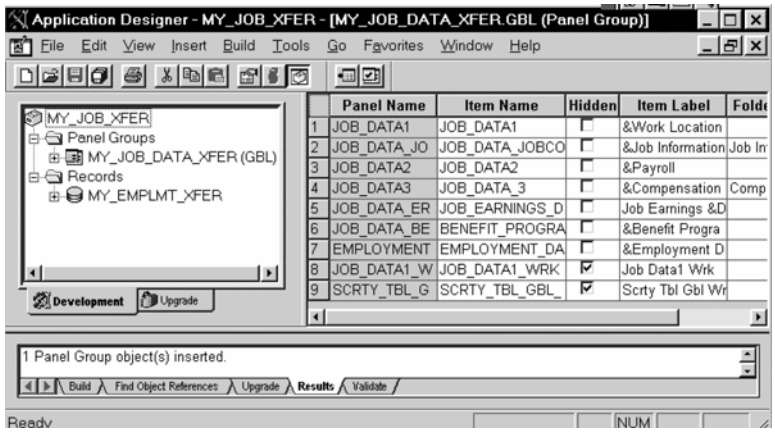


Figure 24.21 A new panel group is added to our project

Even though we added the new panel group to a project, we haven't finished customizing our panel group yet. We just wanted to save our work. Now we'll proceed with the rest of our modifications. Our next step is to modify the panel group

properties. We type some useful description in the General tab, then switch to the Use tab of the Panel Group Properties. Let's specify the search record as MY_EMPLMT_XFER (figure 24.22).

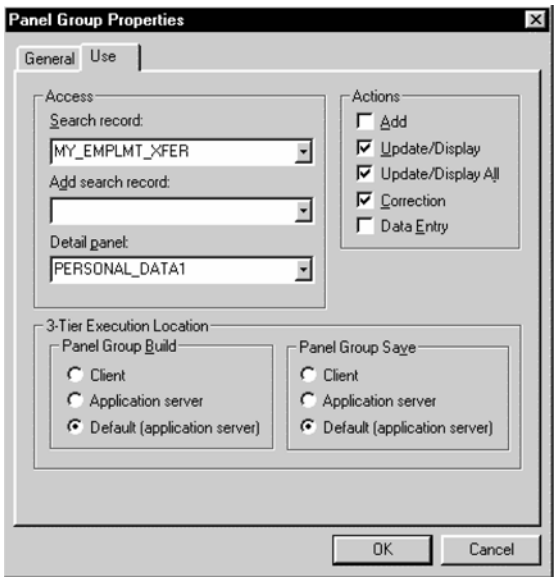


Figure 24.22
Specifying the search record for our custom panel group

Click on the OK button and save our changes. By executing this step, we actually linked the newly created custom search record to the panel group.

We are now ready to modify the menu.

24.4 MODIFYING A MENU

Since our users want the new Job Data Transfer panel group to be accessed from the same menu as the Job Data, we open the ADMINISTER_WORKFORCE_(US) menu (figure 24.23).

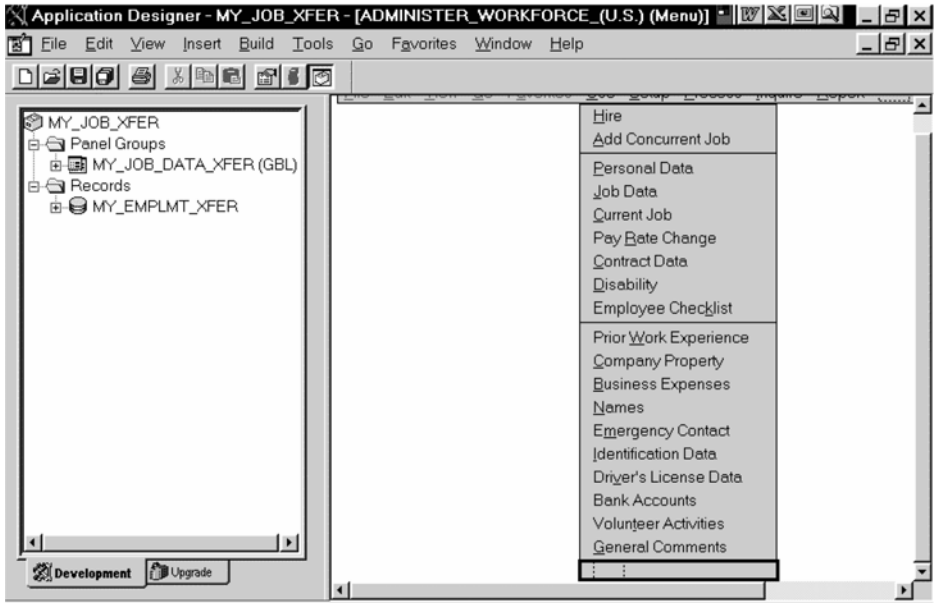


Figure 24.23 Modifying the delivered menu

We double-click on the empty rectangle and specify our own menu item name, a label, and a panel group. This menu should be linked to the MY_JOB_DATA_XFER panel group (figure 24.24).

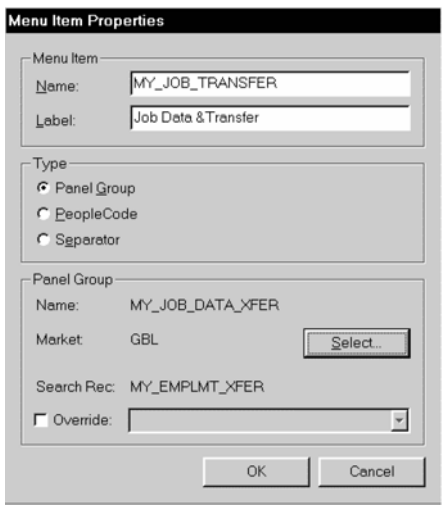


Figure 24.24
Creating a custom menu item

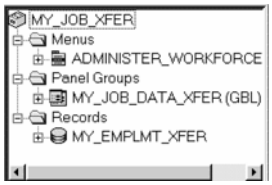


Figure 24.25 Adding the Administer Workforce US menu to the project.

Type in the menu item name and label that will be displayed for users. Select the panel group MY_JOB_DATA_XFER for this menu and specify the MY_EMPLMT_XFER as a search record. Let's move the new menu item next to the delivered Job Data menu item by dragging and dropping the object. After completing the menu changes, we save it. Our project is shown in figure 24.25.

24.5 GRANTING SECURITY ACCESS

Navigation: Go →PeopleTools →Security Administrator →Open →ALLPANLS →Menu Items

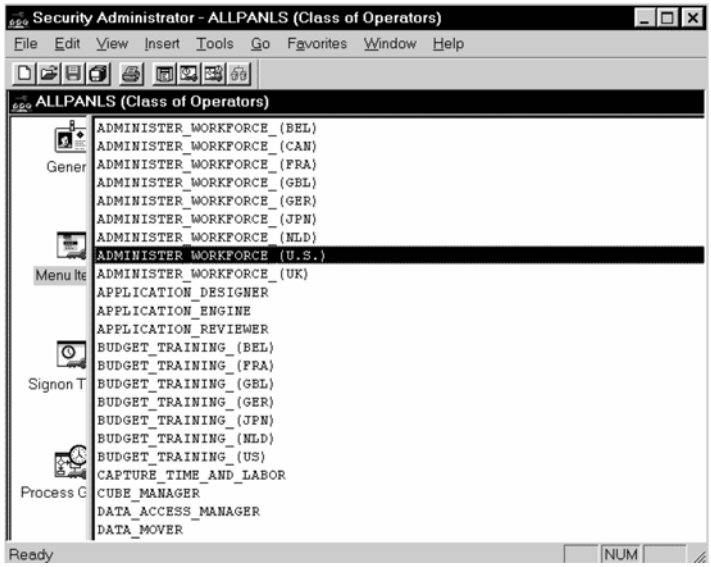


Figure 24.26 Selecting the Administer Workforce (U.S.) menu

After selecting the Administer Workforce (U.S.) menu, the security panel looks as shown in figure 24.27

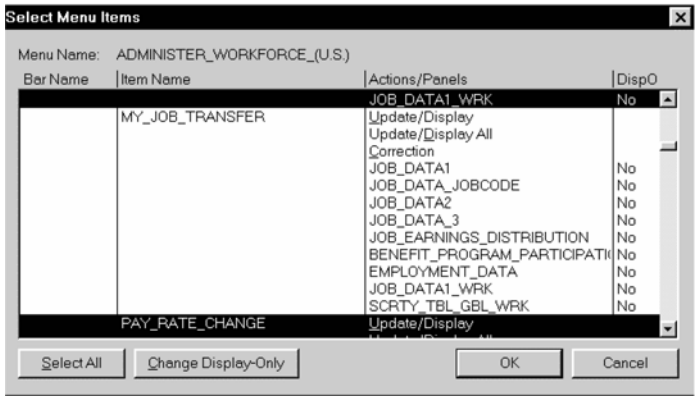


Figure 24.27 Granting user's security to the Job Transfer menu item

Highlight every line that belongs to the MY_JOB_TRANSFER Menu Item and then click the OK button.

24.6 TESTING OUR CHANGES

Before we start testing, let's state the expected results. Our goal was to allow end users access to their transferred employee's records. In other words, if an employee were transferred to another department, and if we have no access to the employee's new department records, we should still be able to see the transferred employee records using our newly created objects. At the same time, if we try to access the same records via the regular PeopleSoft-delivered Job Data panel group, the employee records will not be available.

At first, let's examine the department security access for the ALLPANLS operator class, since this is the class we use for our testing purposes (figure 24.28).

After selecting the ALLPANLS operator class and clicking on the OK button, we get a list of departments with their respective access codes (figure 24.29).

Navigation: Go →Define Business Rules →Administer HR System →Use → Maintain Data Security

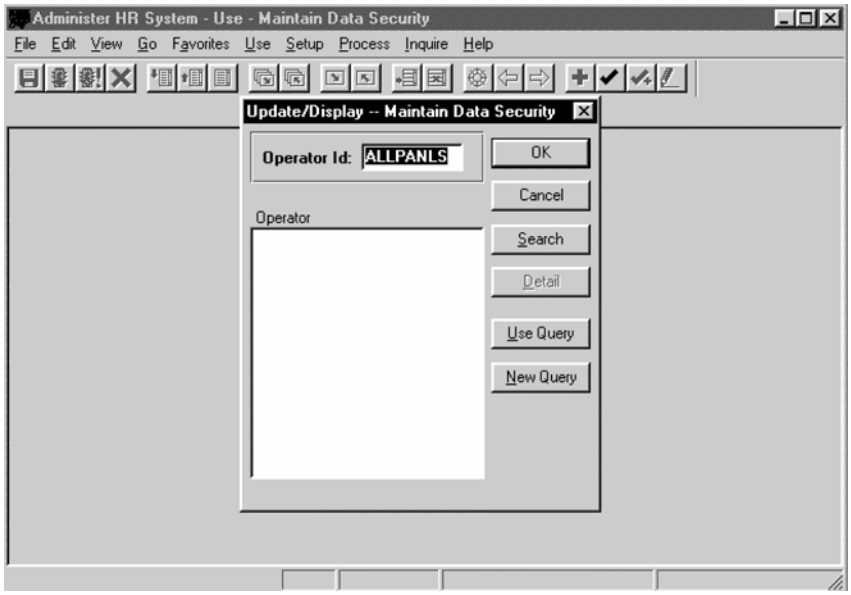


Figure 24.28 Selecting the ALLPANLS operator class

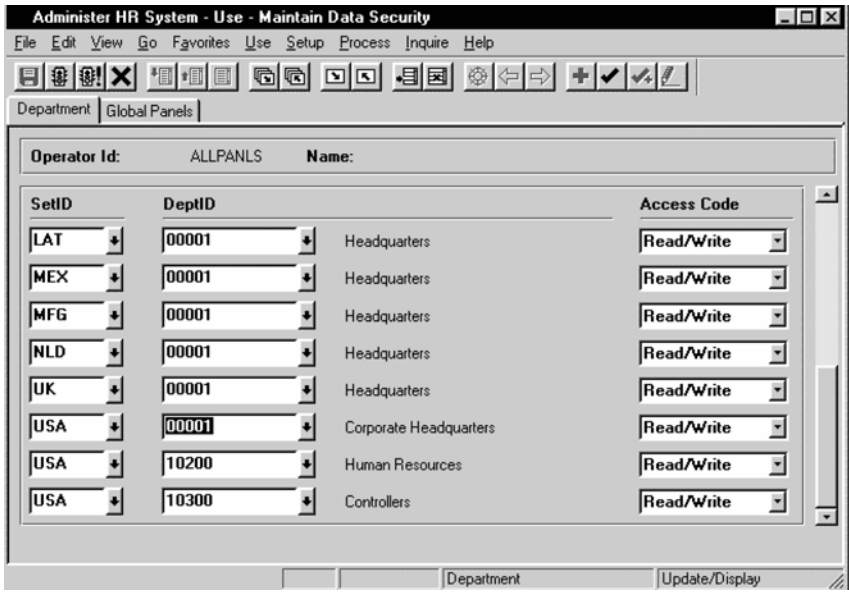


Figure 24.29 Department security for the ALLPANLS operator class

For our test purposes, let's modify the ALLPANLS operator class and deny access to one of the departments—for example, department 10200, Human Resources—as shown in figure 24.30.

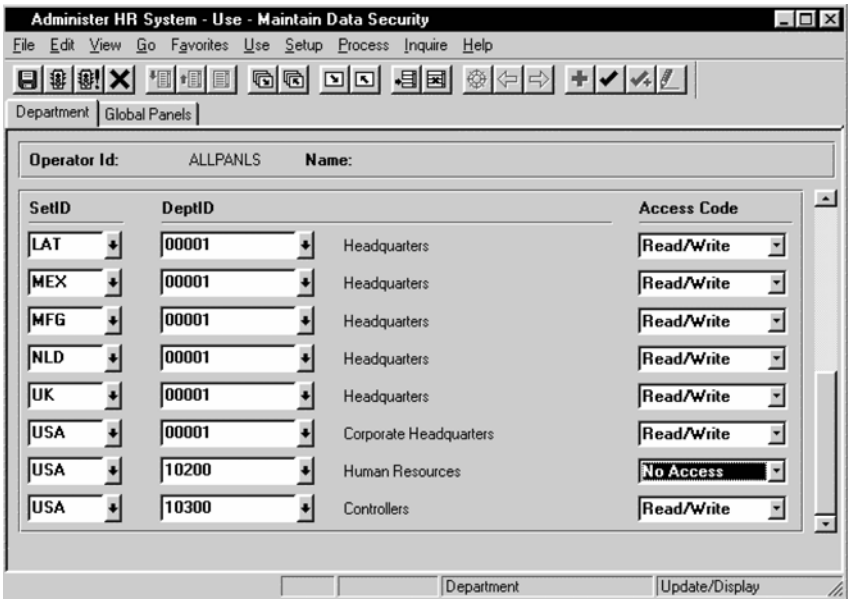


Figure 24.30 Changing the access code for the Human Resources Department to No Access

Our next step is to transfer an employee to department 10200, Human Resources.

Navigation: GO →Administer Workforce(U.S.) →Use →Job Data →Work Location → Update/Display All

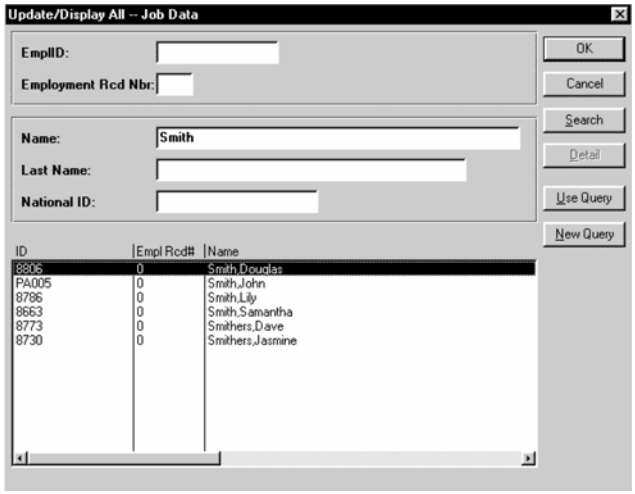


Figure 24.31
Selecting an employee to be transferred to another department

After selecting "Smith, Douglas," the first employee in the list, we transfer this employee to department 10200 by inserting a new row (figure 24.32).

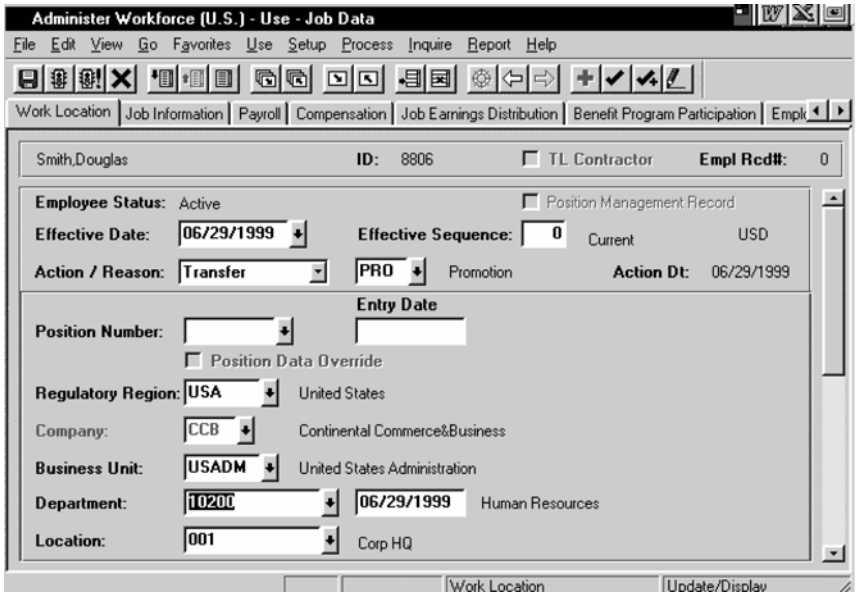


Figure 24.32 Transferring Smith Douglas to department 10200

After saving the record, we can try to access it again. Let's enter "Smith" again as a partial key in the Job Data search box.

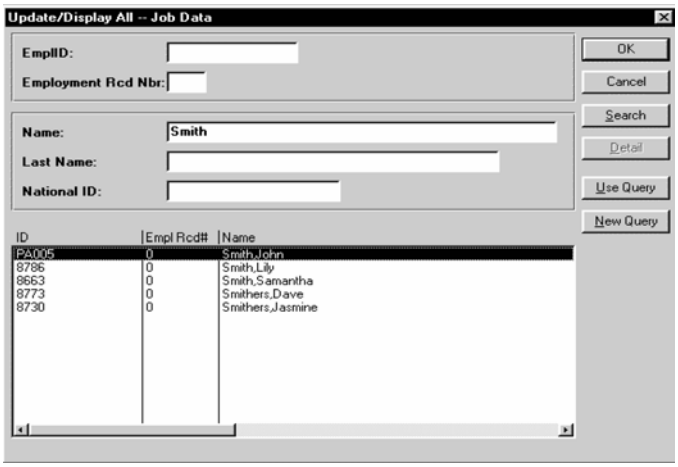


Figure 24.33
Selecting the records of all employees with last names starting with "Smith…"

As you can see from our selection list, the record of "Smith, Douglas" is not found. Of course, we know the reason. This employee has been transferred to a department to which our operator class does not have security access.

Now, let's see how our modifications work. We select the same employee, but this time from our new menu item.

Navigation: GO →Administer Workforce(U.S.) →Use →Job Data Transfer → Work Location →Update/Display All

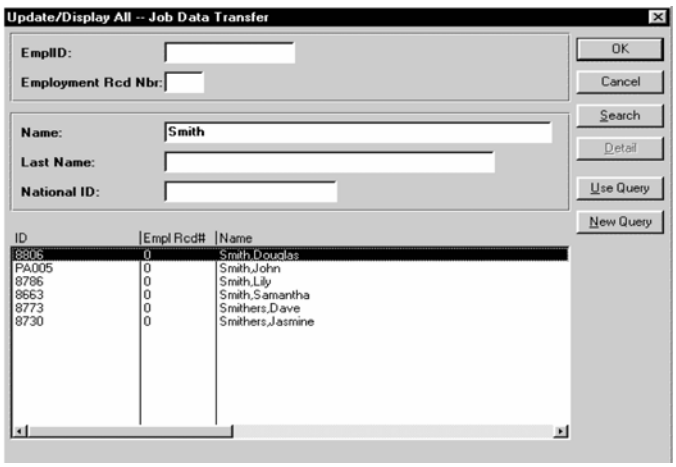


Figure 24.34
Selecting all Smith's records from the Job Data Transfer menu

The record of "Smith, Douglas" is in the list. This means that our new security search record, which is solely responsible for selecting items in the list, is working. Let's select this employee and verify his records (figure 24.35).

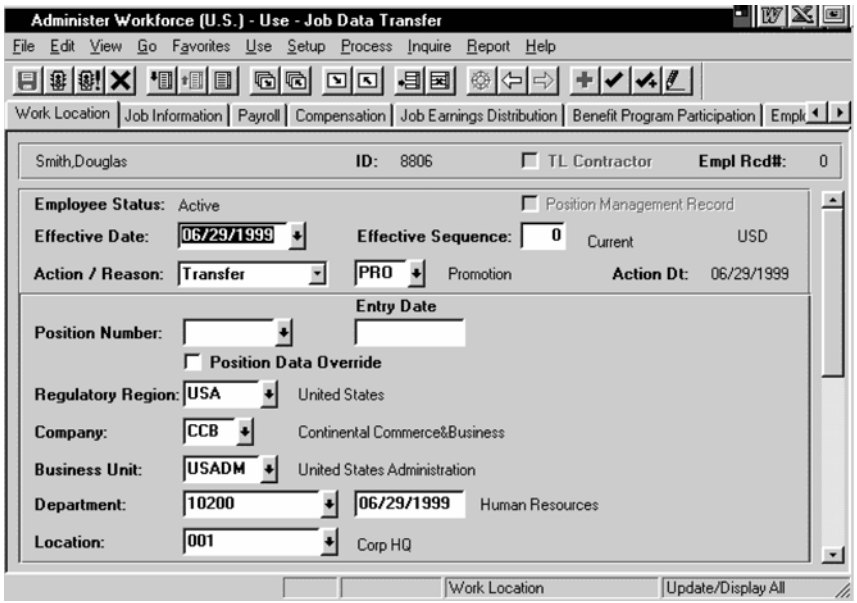


Figure 24.35 "Smith, Douglas" record selected from the new menu item

Let's recall that our users should have access to the records of their former employees, but they should not see the compensation information for the departments to which they don't have access. Switch to the compensation tab to see if this requirement is met (figure 24.36).

We allowed our user to access the records of his/her former employee, but at the same time we compromised the security itself. According to the security that we set up earlier, our user should not have access to the Department 10200 and, therefore, should not see the sensitive information for this department. To resolve the problem, we can write a PeopleCode program to either hide the entire records for the department to which the user does not have access or hide all sensitive fields in these records.

We have to go back to the drawing board (which often happens in the real life development), talk to our users again, present them with some options, and decide on the strategy. Let's suppose that our users decided to go with the second option—i.e., to allow access to all the records, but hide the sensitive salary information in the departments to which users do not have access.

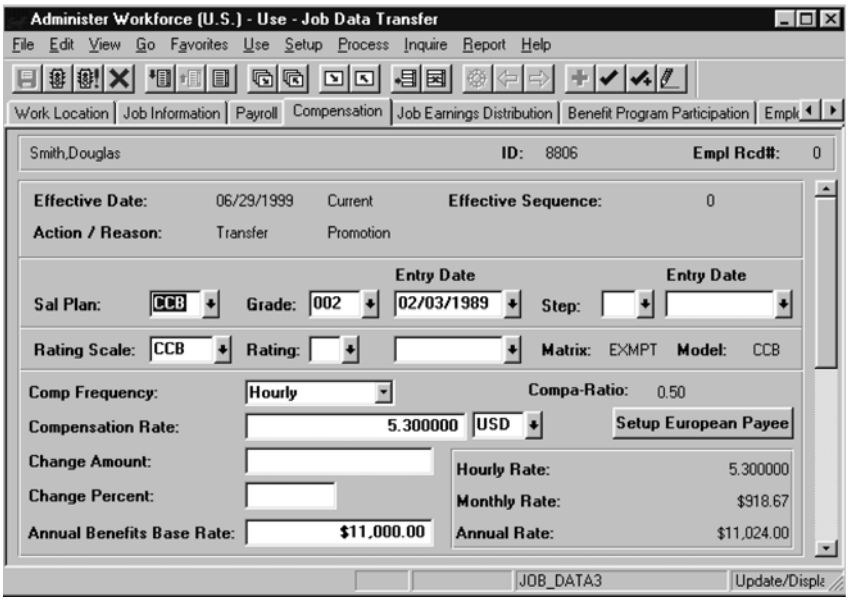


Figure 24.36 The Compensation Tab for Douglas Smith's record contains all the employee's salary-related information

24.7 DEVELOPING A PEOPLECODE PROGRAM

Our goal is to create a PeopleCode program that enables us to hide salary information in the JOB records to which users do not have access. How do we know which user has access to a particular department? Our knowledge about the Department security that we gained during our trace activity at the beginning of this chapter will help us to do this. When an end user enters the search key (full or partial) for the employee record, the Application Processor retrieves all matching records into the buffer based on the selection criteria and our redesigned search record. Then, before the records are displayed on the screen, all PeopleCode programs from the `FieldDefault`, `RowInit`, and `RowSelect` events are fired for every record behind the panels in the panel group. If we want to display only the records to which our user has access and discard the others, the `RowSelect` event with a DiscardRow command will help us make the Application Processor skip the current row of data and continue processing other rows. Since our task is to hide certain sensitive fields, we use another PeopleCode event, the `RowInit` event. The `RowInit` programs are responsible for setting up the initial display of data.

Now that we've decided in what event to place our code, we need to think of how to best achieve this. Our program should analyze the operator class that accesses the panel group, then, for each selected employee record, get the department code, and verify if the operator class has access to this department. In order to do so, we need

to create a PeopleCode program with SQLExec statements similar to the SQL in the search record. When dealing with SQL statements in PeopleCode, it is always a good idea to create your SQL statements outside of PeopleCode and test them using your database manipulation tools. Take a look at the security view SQL at the beginning of this chapter. Let's just copy a portion of the SQL that deals with department security. This time we will look at the lower portion of this view:

```
AND SEC.ACCESS_CD = 'Y'
AND EXISTS
   (SELECT 'X'
    FROM PSTREENODE SEC3
    WHERE SEC3.SETID   = SEC.SETID
    AND SEC3.SETID     = B.SETID_DEPT
    AND SEC3.TREE_NAME = 'DEPT_SECURITY'
    AND SEC3.EFFDT     = SEC.TREE_EFFDT
    AND SEC3.TREE_NODE = B.DEPTID
    AND SEC3.TREE_NODE_NUM BETWEEN
        SEC.TREE_NODE_NUM AND SEC.TREE_NODE_NUM_END
    AND NOT EXISTS
        (SELECT 'X'
         FROM PS_SCRTY_TBL_DEPT SEC2
         WHERE SEC.OPRID  = SEC2.OPRID
         AND   SEC.SETID  = SEC2.SETID
         AND   SEC.TREE_NODE_NUM <> SEC2.TREE_NODE_NUM
         AND   SEC3.TREE_NODE_NUM BETWEEN
               SEC2.TREE_NODE_NUM AND SEC2.TREE_NODE_NUM_END
         AND   SEC2.TREE_NODE_NUM BETWEEN
               SEC.TREE_NODE_NUM AND SEC.TREE_NODE_NUM_END
         )
  )
```

Note, that the portion of SQL enclosed in parenthesis uses columns from the PS_JOB table (with the table name alias B) such as SETID_DEPT and DEPTID and from the SCRTY_TBL_DEPT (with the table name alias SEC) such as SETID and TREE_EFFDT. For testing purposes, we can plug in specific values relevant to our test employee ("Smith, Douglas") into this SQL code. Let's put our SQL together and execute it (figure 24.37).

As you can see, we plugged in the following values: the OPRID='ALLPANLS', SETID='USA', TREE_NODE='10200'. Using these values, our SQL did not return any rows (figure 24.38). This is exactly what we expected, since we turned off access to this department earlier in our test. Just to make sure that our SQL is working correctly, let's plug in another department—this time the one to which the ALLPANLS operator class has access—and run it again.

Our test SQL returned one row. We now know that our SQL is working correctly.

The next step is to decide where we should place our RowInit PeopleCode event. We also need to know the names of the fields we need to hide. First, let's find the panel name where the salary-related information for the employee must be hidden.

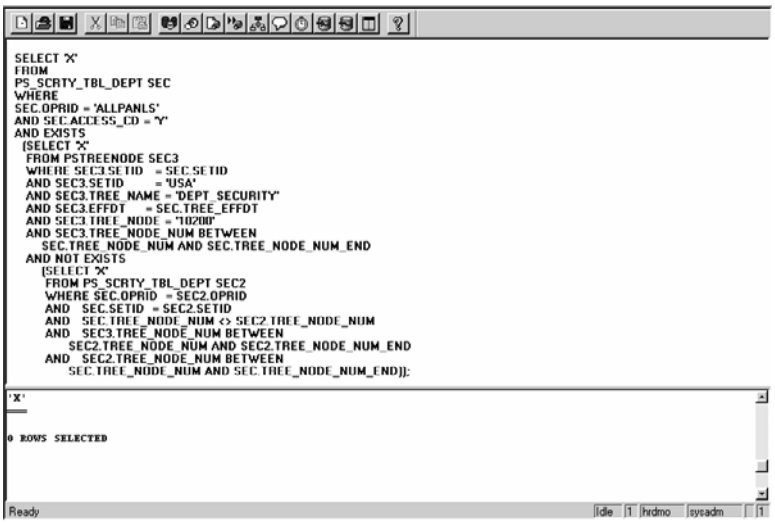


Figure 24.37 Executing the SQL statements outside PeopleSoft for testing purposes

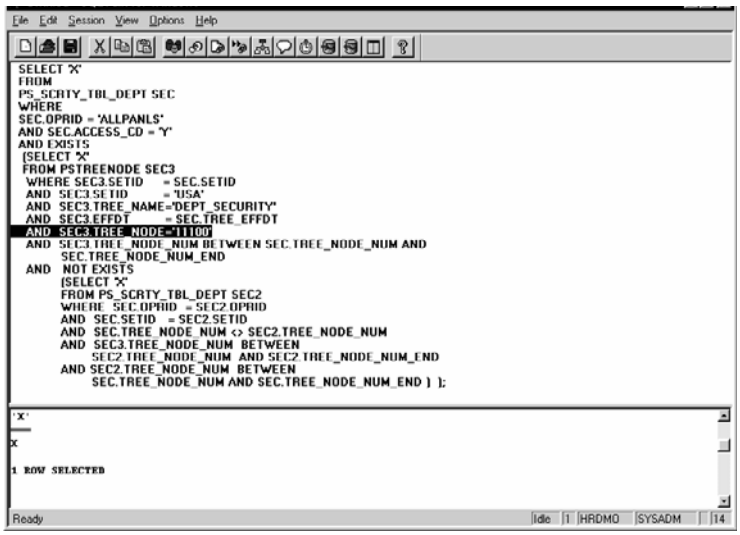


Figure 24.38 Testing SQL with department 11100

If you look back at figure 24.36, you can see that the Compensation tab panel name is JOB_DATA3.

Now we can open the JOB_DATA3 panel and determine the names of all the fields we are planning to hide. After the panel is opened, let's examine the Order panel (figure 24.39).

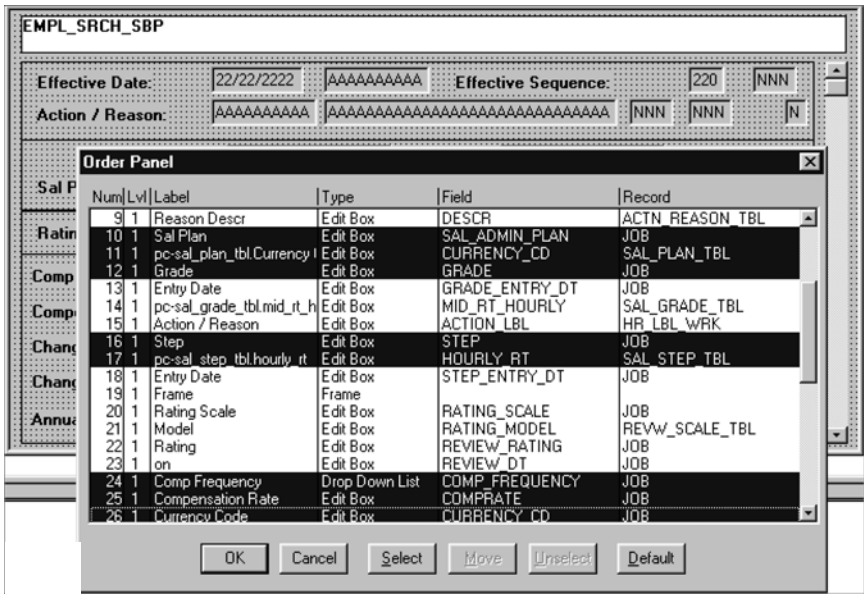


Figure 24.39 Examining the Order Panel for the JOB_DATA3 Panel

As you can see from figure 24.39, all salary-related fields belong to the JOB record. These fields need to be hidden in our customized panel. Let's place the PeopleCode script that hides the salary-related fields to the JOB.EMPLID `RowInit` event. In order to minimize the necessary customizations to PeopleSoft-delivered PeopleCode, let's create a function outside the JOB record.

PeopleSoft, by convention, usually places its functions in the `FieldFormula` event of the derived FUNCLIB records. These records (FUNCLIB_HR, FUNCLIB_BEN, FUNCLIB_PAY, and so forth.) are used as placeholders for the external functions called from different record field events. Generally, a function may belong to any record event. (Please see part 3 of this book for more details about function libraries.) We'll create our own derived record here to hold our first function, as well as any others that may follow.

24.7.1 Creating a derived Funclib record and PeopleCode

Let's create a record named MY_FUNCLIB and add one field to it. Let's re-use the existing EMPLID field. Remember, the purpose of this record is just to hold PeopleCode functions.

Before the record is saved, do not forget to change its properties, add some useful comments, and define this record as a Derived/Work record. We don't need to use the Build option, since this record is not going to be created at the database level.

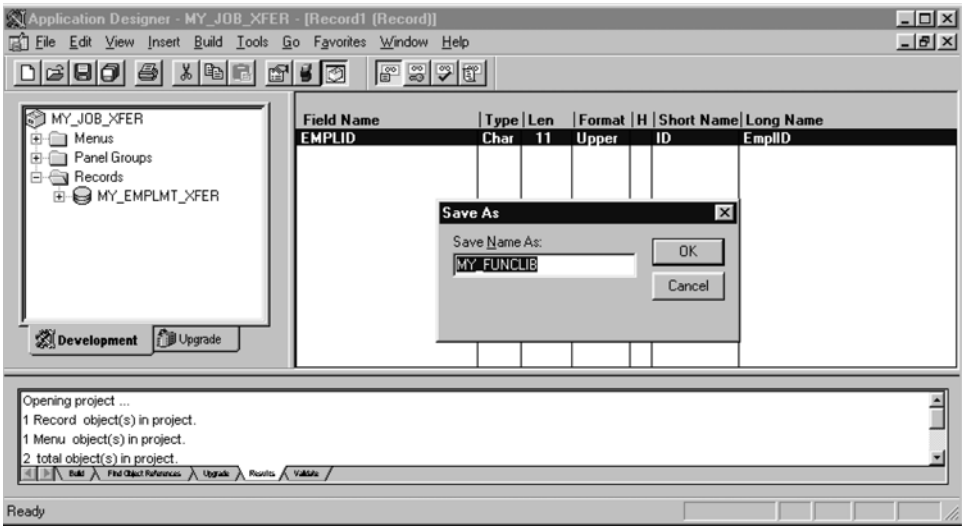


Figure 24.40 Creating a derived record for our custom function library

Now we can start by creating a function named MY_CHECK_SECURITY. This function will control the user's access to departments for each JOB record in the buffer. We place this function in our newly created MY_FUNCLIB derived record, EMPLID FieldFormula event (figure 24.41).

Since our PeopleCode function has grown quite big, let's display it separately in order to understand how it works:

```
Function my_check_security(&SETID_DEPT, &DEPTID);
   &OPERATOR = %OperatorClass;
   SQLExec("SELECT 'X' FROM PS_SCRTY_TBL_DEPT SEC WHERE SEC.OPRID=:1 AND
SEC.ACCESS_CD='Y' AND EXISTS (SELECT 'X' FROM PSTREENODE SEC3 WHERE
SEC3.SETID  = SEC.SETID AND SEC3.SETID  = :2 AND
SEC3.TREE_NAME='DEPT_SECURITY' AND SEC3.EFFDT= SEC.TREE_EFFDT AND
SEC3.TREE_NODE=:3 AND SEC3.TREE_NODE_NUM BETWEEN SEC.TREE_NODE_NUM AND
SEC.TREE_NODE_NUM_END AND NOT EXISTS (SELECT 'X' FROM PS_SCRTY_TBL_DEPT
SEC2 WHERE SEC.OPRID = SEC2.OPRID AND SEC.SETID = SEC2.SETID AND
SEC.TREE_NODE_NUM <> SEC2.TREE_NODE_NUM AND SEC3.TREE_NODE_NUM BETWEEN
SEC2.TREE_NODE_NUM AND SEC2.TREE_NODE_NUM_END AND SEC2.TREE_NODE_NUM
BETWEEN SEC.TREE_NODE_NUM AND SEC.TREE_NODE_NUM_END))", &OPERATOR,
&SETID_DEPT, &DEPTID, &SELECTED);
   If None(&SELECTED) Then
```